INTELLIGENT CRT TERMINAL FOR MICROPROCESSOR LABORATORY

by

KAMAL NAYAN

DEPARTMENT OF ELECTRICAL ENGINEERING

ıIAN INSTITUTE OF TECHNOLOGY KANPUR

DECEMBER, 1981

INTELLIGENT CRT TERMINAL FOR MICROPROCESSOR LABORATORY

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

KAMAL NAYAN

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

DECEMBER, 1981

To

My Parents

EE- 1981-M-NAY-INT

16.12.81
[illegible]

CERTIFICATE

Certified that the work entitled "INTELLIGENT CRT TERMINAL FOR MICROPROCESSOR LAB" by Mr. Kamal Nayan, has been carried out under my supervision and the work has not been submitted elsewhere for a degree.

[illegible]

(R.N. Biswas)
Professor
Department of Electrical Engineering
Indian Institute of Technology
Kanpur-16.

December, 1981

ACKNOWLEDGEMENTS

It was my privilage to work under Dr. R.N. Biswas and I, hereby, wish to express my sincere gratitude to him for rendering immense help and valuable suggestions throughout the project.

I am highly grateful to my friends - Mr. P.C. Pandey, for his invaluable help and encouragement at some very critical moments during hardware testing, Mr. M.V. Govinda, for providing me a memorable 'company', even late in nights, in software debugging, Mr. V.P. Deshmukh, Mr. Pukhraj Kachwaha Mr. Karnal Singh and Mr. D.V. Rama Rao for some suggestions, Mr. Kamal Kumar for burning EPROM. Thanks are due to my brother, Raj Kumar, who took pains to draw flow-charts and some circuit diagrams.

I express my thanks to Mr. B.V. Ramana and Mr. Prem Malhotra (REs) for helping me a lot to make it a success.

Finally I thank all of those who directly or indirectly made my stay comfortable.

I am thankful to Mr. Anil Kaul and Mr. J.S. Rawat for the neat typing work.

[illegible]
Kamal Nayan

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

An attempt has been made to design an intelligent CRT terminal for microprocessor labs aiming at an exposure to microprocessors at a basic level. The motivation for the project is the need for standalone terminals which could function without the help of a computer. The essential features of such an intelligent terminal would be:

i) Keying-in a program and displaying it on a CRT screen

ii) Editing of the program to rectify the mistakes

iii) Execution of a program.

The third feature requires two additional facilities to e provided in the system:

i) An ASCII-to-Hex conversion routine, which will allow the user to enter a machine-language program through the keyboard (which codes each character in ASCII) and then to generate the actual machine codes.

ii) A Hex-to-ASCII conversion routine, which will allow the user to display and examine an object program already stored in the memory.

In addition, a serial communication interface has been ncorporated in the terminal so as to facilitate the transfer of oftware from the terminal to any other device and vice-versa.

The popular 8-bit microprocessor INTEL 8085 has been chosen because of the availability of versatile support chips of the same family. Taking into consideration the need of high data transfer from memory to display circuit, a Programmable Direct Memory Access Controller (8257) has been used. To control the display, a Programmable CRT Controller (8275) has been used which gives the flexibility of changing the display format on the CRT screen. This flexibility has been retained by providing a facility of changing various parameters of associated I/O devices through the keyboard. For serial communication, a Universal Synchronous/Asynchronous Receiver/Transmitter (8251) has been used in the asynchronous mode.

CHAPTER 1

INTRODUCTION

A CRT terminal is an indispensible part of any microprocessor-based system. It is the device with which a user interacts with the system. The requirements of the user are supplied to the CPU through the keyboard of the terminal while the response of the system is communicated to the user through the CRT screen of the terminal. For example in a control application the user supplies various specifications to the microprocessor-based controller through the keyboard and the system apprises the user of its response by displaying it on the CRT screen. The user may change these specifications on the basis of the response to make the system adaptive. Thus CRT terminals play an important role in any microprocessor-based system.

The simplest and the most elementary form of a CRT terminal which one could visualise is a dumb terminal which can only be used in a system having a central computer. Such terminals don't have their own CPUs and as such don't have any decision-making capability. Any key pressed on the keyboard of such a terminal is directly transmitted to the CPU which issues an appropriate command to the terminal regarding what to do. This concept is useful either if the number of terminals required to serve the various users is large or if the task to be done is so complicated that a central computer becomes inevitable.

But on certain occasions the work is not highly complicated. In such a case the use of a central computer which is costlier because of its high capability is not advisable. But instead, terminals which have their own CPUs, though less powerful, are preferred. This CPU is capable enough to handle certaintasks normally required of a terminal. As these terminals do this work on their own, they are known as intelligent terminals.

The main objective of this project is to design a low cost terminal suitable for use in an academic environment where the main motive is to impart microprocessor knowledge through organized laboratory work. As the programs to be run at this level would not be highly complicated, the selection of a dumb terminal with a central computer is not advisable. Instead, an intelligent terminal with the basic facilities of text editing and inter-station communication as well as the capability of program execution/debugging would be more useful. Hence the project is aimed at the design of such a low cost intelligent terminal.

Chapter 2 describes the various features desired in an intelligent terminal while chapter 3 has been devoted to the system architecture giving all hardware details. In chapter 4 the development of various software routines has been discussed in details. The thesis is concluded with a discussion on the limitations of the design and the possible improvements in chapter 5.

CHAPTER - 2

FEATURES OF TERMINALS

This chapter deals with the various features of an intelligent terminal desirable for the present project. The features such as text editing, inter-station communication, program execution/debugging, LSI testing and EPROM programming are discussed in details in the following sections.

2.1 EDITING OF TEXT

It is the most elementary but significant feature of an intelligent terminal. The user enters his program through the keyboard of the terminal which is stored by the system in its RAM area. The contents of the RAM are displayed on CRT screen which enables the user to see his text. In case of any mistake he can rectify it by using the text-editor stored in the system ROM. The various editing facilities given by such a text editor are listed below :

(1) Cursor movements-up,down,right,left,home
(2) Deletion or insertion of a character
(3) Deletion or insertion of a line
(4) Scrolling-up or down
(5) Carriage return
(6) Provision of protected region.

2.2 INTER-COMMUNICATION WITH OTHER WORKSTATIONS

In a microprocessor-based system, inter-communication with similar type of work-stations is highly desired so that the users at different terminals may be benefitted by an easy transfer of software. This communication may be serial or parallel depending upon the speed desired and the distances involved; For a high speed requirement parallel communication is the only choice while serial communication is less expensive. Thus a trade-of is to be established between speed and cost. If distances to be covered are large, one can associate MODEMS and use telephone lines for serial communication. Several manufacturers have come out with LSIs which result in serial or parallel communication e.g. USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial communication and various standard bus interfaces like GPIB for parallel communication.

2.3 PROGRAM EXECUTION

If the CPU provided in the terminal is just confined to text-editing or inter-station communication, it won't be fully utilized, in view of the usual capability of such a CPU. Moreover, for program execution, which is quite common in any microprocessor based system, the user would have to look for another CPU if this facility is not provided. Hence, if such a feature is added to a terminal, it becomes more versatile.

A CPU can execute only those programs, which are written in its machine language. But as programming in machine language is quite cumbersome, one normally uses an assembly language, which consists of mnemonics. The only overhead associated with this is the need of an assembler which converts the assembly language program to executable machine code. One more step ahead is the concept of higher level language programming in which one instruction may correspond to many machine language instructions unlike the assembly language programs where one instruction corresponds to one machine instruction. Higher level language programs are converted by a compiler to the machine code of CPU.

The selection out of these two choices entirely depends on need and program size. If the programs are too lengthy, higher level language is preferred which reduces programming efforts considerably. But in such a case a large memory is required to store the compiler. Moreover, as the object code produced by a compiler is always lengthier than the object code written in assembly language or machine language, the memory requirement is further enhanced. This also results in more execution time than that of assembly/ machine language programs. Assembly language programming is therefore preferred if programs are not highly complicated or they are to be run on line.

In the applications where this terminal is envisaged to be used, program size is not expected to be big enough to warrant the need of a compiler. Hence it is assumed that the user would enter and edit his assembly language program through the keyboard. For execution, this program would have to be converted to object code by an assembler. But as the object code would not be displayable, it is better if a facility to convert this executable code to displayable code is provided. This would help the user to check the assembled code. This requires a software routine which converts this code into a suitable format for display in the hex, octal or binary code. Occasionally a user prefers to write his program directly in machine language through the terminal. This facility can be incorporated simply by having a software routine for converting the keyboard code (ASCII) to the desired object code.

2.4 DEBUGGING AID

This is a powerful and a desired feature of an intelligent terminal. It helps the user in trouble-shooting as the user can actually see the program flow very clearly. In an environment where education of microprocessors is to be imparted, the usefulness of this feature is all the more significant because learners can thereby understand how the system executes a program. There are two modes of debugging :

(1) Single instruction execution

(2) Break point

In the first case the CPU executes one instruction and then stores the contents of latched address bus, latched data bus, and its various registers into some memory locations from where it is displayed. After this it executes the next instruction and then again stores the various contents described above, i.e. after the execution of each instruction the status of various lines is stored and displayed on the screen. While in the later case CPU does'nt execute only one instruction at a time but goes on executing a number of instructions until it hits a break point (set already). A break point may be defined as an address, data or some register content. Upon detecting a break point, the CPU terminates the execution of this program temporarily and jumps to a routine which, as discussed above, stores the status of various lines in RAM. After executing this routine the CPU restarts the execution of old program as if nothing happend. Here again a break point can be set. Thus the first mode is the thorough check-up of program flow while the second mode is the random check-up of program flow but may be desired than the first one in certain cases. The status of various lines thus stored will again not be displayable which needs an interpreter to convert it into displayable one.

2.5 LSI TESTING

Any microprocessor-based system incorporates many LSIs; hence the need of testing such LSIs is obvious. As these chips are programmable by microprocessors, their testing is entirely different from the testing of SSI's and MSI's. Some appropriate commands are to be given at some suitable timings to check the behaviour of such chips. As an intelligent terminal has its own CPU, it can be exploited to perform this task. In fact, it can be done easily with such a terminal simply by adding a little more software.

The LSI to be tested can be connected directly to the data bus of such a system. The CPU may then be asked to execute a small routine, written for testing this chip, which sends some commands, parameters or data to this chip and then it may sample the output to verify the functioning of this chip. This output can again be stored in RAM (not displayable) and displayed on CRT screen by invoking the interpreter mentioned in 2.4. This gives the user a full facility to analyze the operation of such chips.

2.6 EPROM PROGRAMMING

This capability is in fact just a special case of LSI testing. In this case the user enters his program through the keyboard, which is converted to the machine code by the interpreter or assembler as the case may be. Then the CPU executes a program which copies this RAM area (where machine code is stored) into specified ROM area which is known as the programming of an EPROM. In certain cases, a need may arise for modifying only certain locations of a ROM. For this the whole ROM is copied into a specified RAM area and then the user changes the contents of the locations to be modified. The EPROM can then be reprogrammed to get the desired result.

2.7 DESIRED SYSTEM SPECIFICATIONS

In view of the foregoing considerations, the basic features to be incorporated in the system may be listed as follows :

(a) Complete text editing facility,

(b) Serial data transmission/reception,

(c) Execution of resident machine language programs,

(d) Built-in HEX-ASCII and ASCII-HEX conversion.

CHAPTER - 3

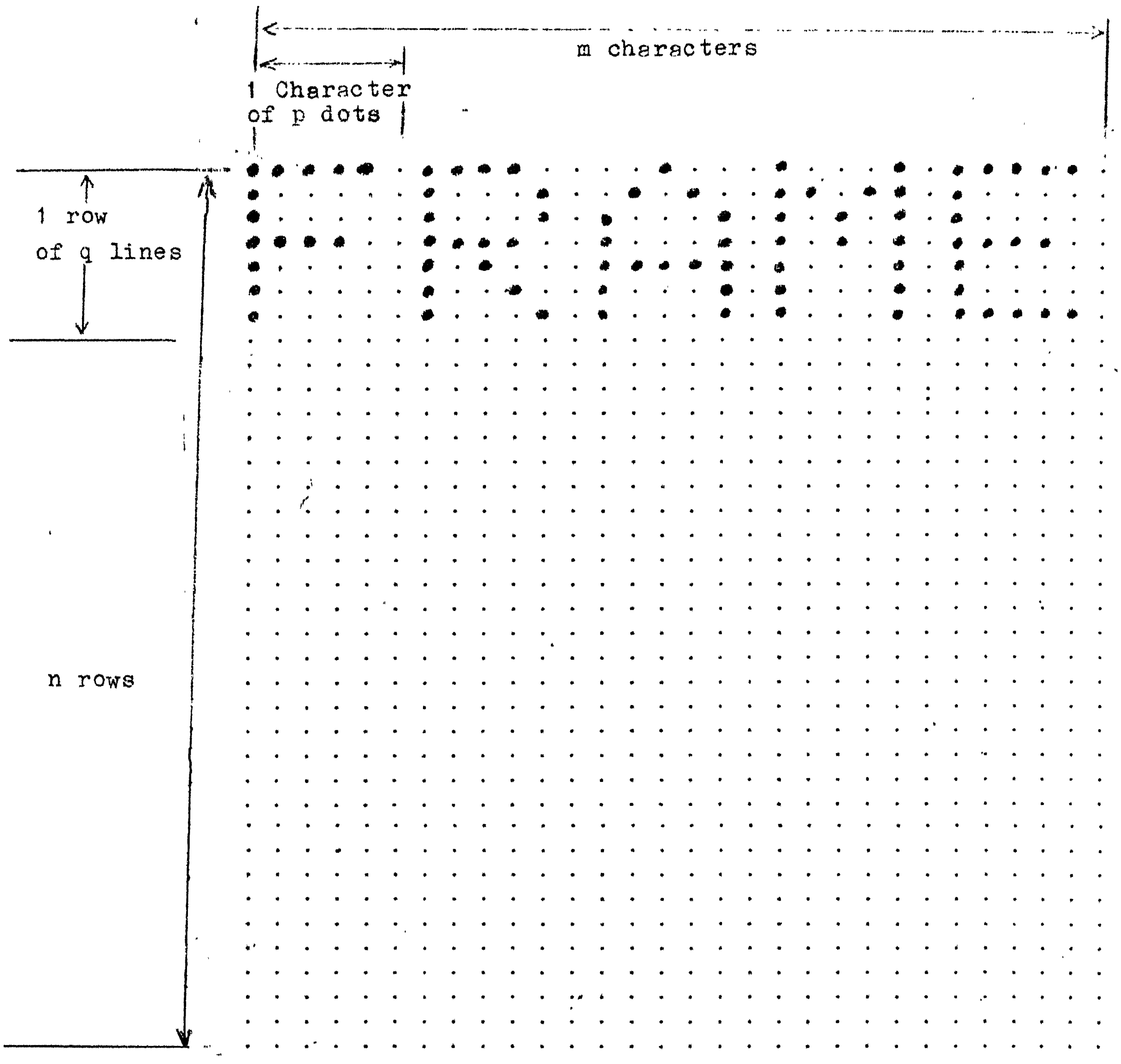
SYSTEM ARCHITECTURE

This chapter is devoted to the system architecture at chip levels. Various hardware details starting from Video generation to the actual bus configuration and other interfacing details are taken up here. It includes the details of a number of LSIs used in the project, viz the Programmable CRT Controller (8275), Programmable DMA Controller (8257), PPI (8255), Programmable Timer (8253) and universal serial Interface (8251). This chapter ends with a discussion on the addressing of various I/O devices and memory units.

3.1 CRT DISPLAY GENERATION

In the Raster Scan technique, the electron beam scans the frame line by line. In this scanning the intensity of the beam is controlled to give a pattern of dots which are white or dark depending upon the characters to be displayed. Fig. 3.1 shows such a frame consisting of m characters per row and n rows per frame.

After the scan of each line, the beam is to be brought back to the left side. This is accomplished by the Horizontal Synchronizing pulse (HSYNC). The retracing is made invisible by the Horizontal Retrace Signal (HRTC). Similarly once the scan of a full frame



Dot Clock = x MHz

Character clock = x/p MHz

LSB of Line Counter or HRTC $= \frac{1}{(m+m')} \cdot f_{\text{character clock}}$

MSB of Line Counter of LSB of Row Counter $= \frac{1}{q} \cdot f_{HRTC}$

MSB of Row Counter $= \frac{1}{qn} \cdot f_{HRTC}$

VRTC $= \frac{1}{(nq+n')} \cdot f_{HRTC}$

Fig. 3.1: Frame Picture

is over, the electron beam is to be brought back to the top left corner. This is achieved by means of a signal known as Vertical Synchronizing pulse (VSYNC). The signal which blanks the screen at this time is known as Vertical Retrace Signal (VRTC). These various signals are suitably added to give the composite video signal as shown in Fig. 3.2 which is fed to the TV monitor. The relative repetition rates of those signals are given in terms of m, n, p etc. as indicated in Fig. 3.1.

A basic schematic of generating the video signal is shown in Fig. 3.3. As at one time only one line of the character dot matrix is displayed on the screen, these characters are required in the scanning of each line of a row. If these characters are fetched from memory during the scan of each line, it would keep the data bus busy all the time in this task and the system would'nt be usable for other purposes. Instead if the characters to be displayed in one row are fetched from memory just once and stored in some shift register in such a way that the shift register gives the required character in the scan of each line, it would spare the data bus for other purposes. This needs a row buffer whose capacity is equal to the maximum number of characters to be displayed per row. In order that there be no delay between the completion of one row and the commencement of the next, two row buffers have to

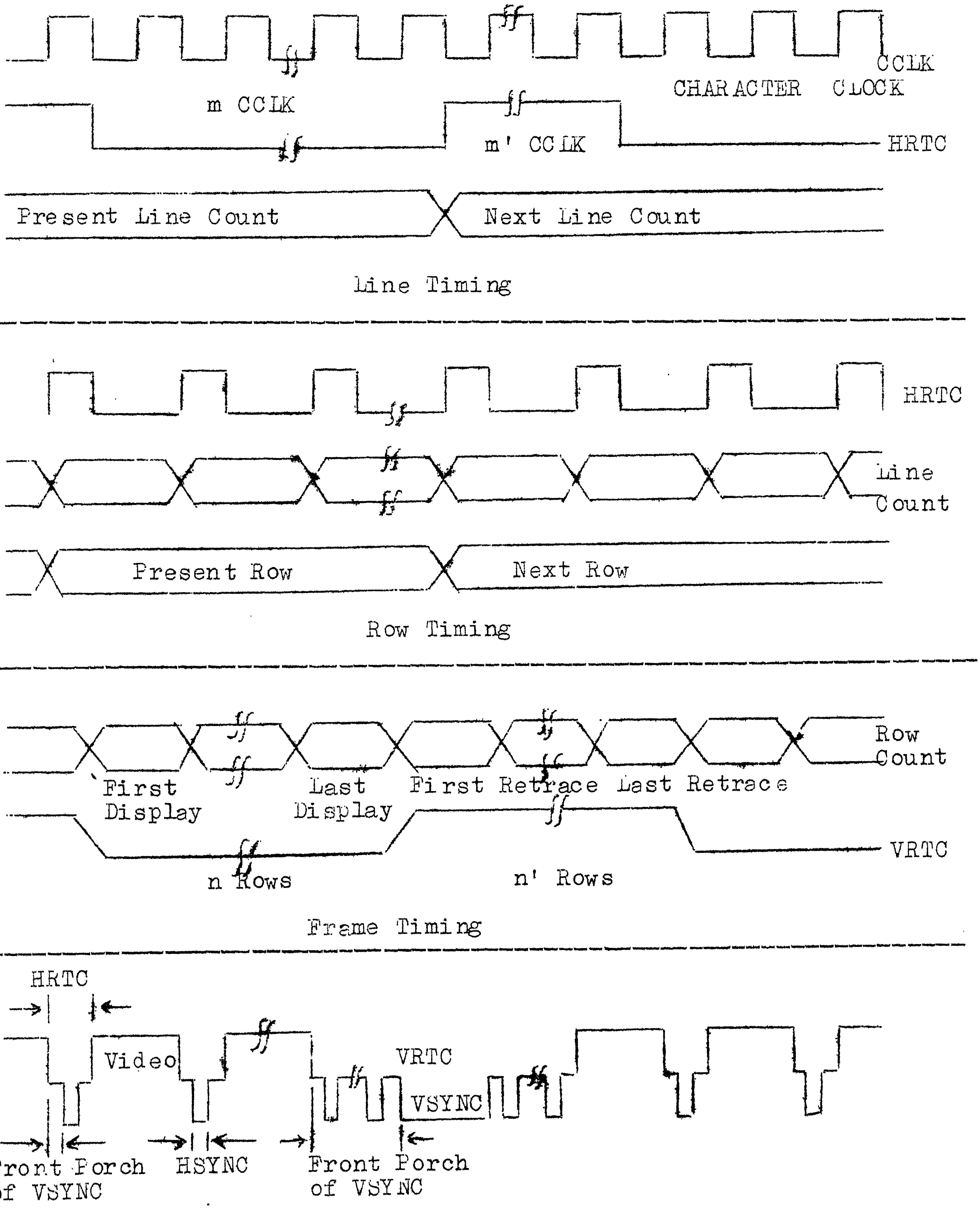


Fig. 3.2: Composite Video Waveforms

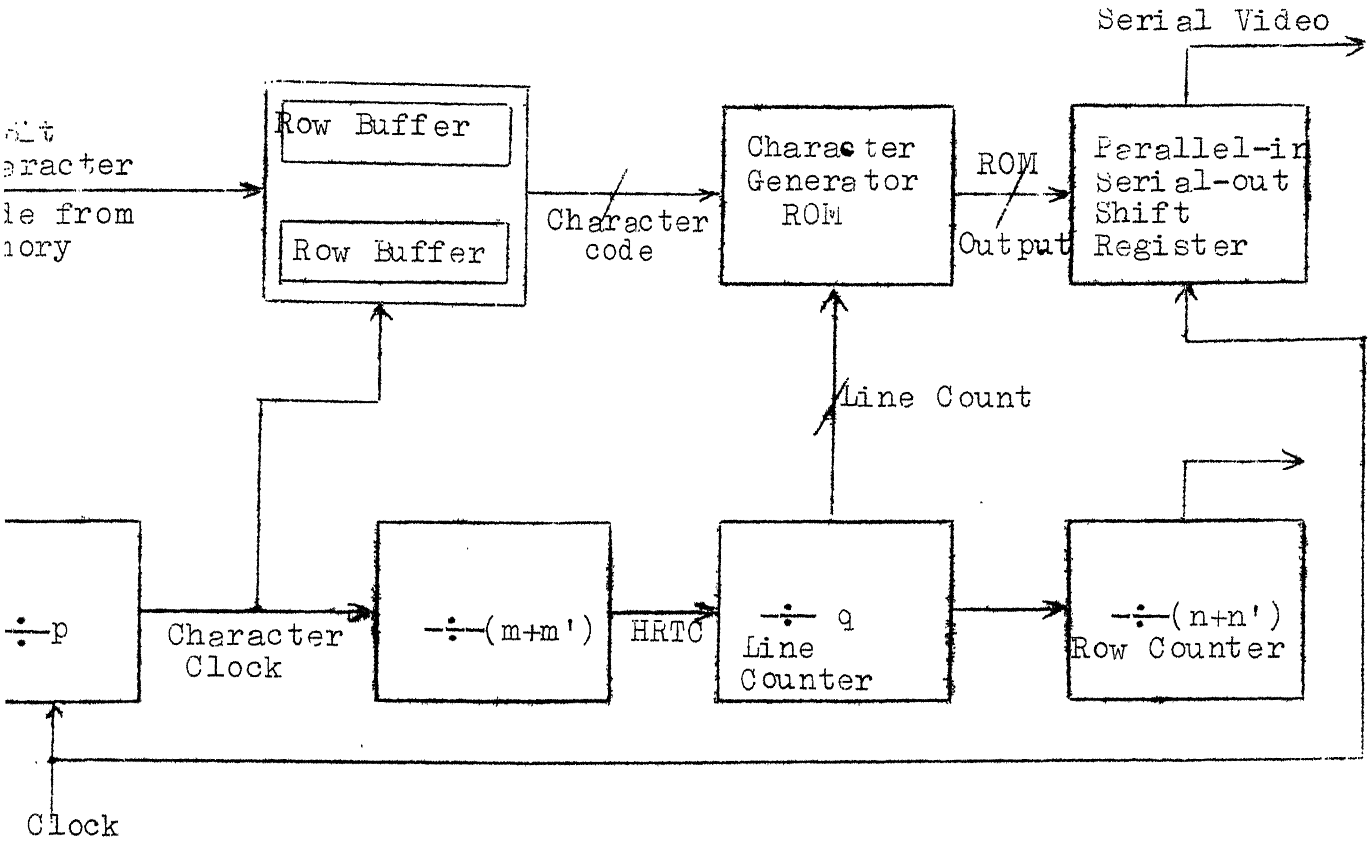


Fig. 3.3: Basic Schematic for Generating Serial Video

be used. While one row buffer is busy in displaying the characters of one row, the other row buffer gets filled by the characters to be displayed in the next row. Moreover, the second buffer must be full before the end of the row being displayed to ensure an uninterrupted display.

The row buffer sending the characters out acts as a shift register clocked by the character clock. Each row requires m character clocks, followed by an HRTC pulse having a duration equal to that of m character clocks. The HRTC pulses are in turn, counted by the line counter. This Line count and the character code supplied by the row buffer together form the address to the character generator ROM which contains the dot patterns of all characters. For each character clock it gives a parallel p bit output corresponding to the selected line of the character to be displayed. As these p dots are to be supplied serially to the video ckt, a parallel-in serial-out shift register is used which takes in p dots supplied by the character generator ROM and sends out p bits serially clocked by the dot clock, which has a frequency p times that of the character clock. This process is repeated for each character clock, untill all the characters in the displayed row buffer are scanned once. In the scanning of the next line the same characters are again fed one by one to Character Generator but as the line count is different, a different code output is fed to the parallel-serial shift register. Thus the successive lines of the characters held by the row buffer are scanned one by one. When one row has been scanned fully i.e. all the lines of one character row have been displayed, the roles of the row buffers are reversed. So now the row buffer which was being filled by the characters from the memory starts displaying them, while the other one which was displaying the characters starts getting filled with the characters to be displayed in the next row. This way the whole frame is scanned.

3.2 PROGRAMMABLE CRT CONTROLLER

The scheme suggested above results in a reasonably complex hardware design which requires a large number of SSI's and MSI's. INTEL has come out with a single LSI programmable CRT Controller (8275) which does most of the functions described above, resulting in a significant saving in component count. This section deals with this chip in detail.

The INTEL 8275 has two 80-character buffers. Their roles are exactly similar as described in the preceding section. During the system initialization, the CPU sends certain commands to this chip which define the exact functioning of 8275. The attractive feature attached with this chip is that it is programmable by software to meet the standards of any TV monitor or the need of the user. All the parameters m, m', n, n', p, q shown in Fig. 3.1 are programmable by software instructions. This chip is used with a Direct Memory Access (DMA) Controller which, upon request from 8275, supplies the characters to be displayed without CPU intervention. 8275 can be programmed to request the DMA in burst transfer in which upto 8 characters can be transferred in one request. The burst length available are 1,2,4 or 8 characters. The interval between bursts is also programmable from 0 to 55 character clocks. Thus a great flexibility has been provided to suit the user's needs. Further the 8275 can be programmed to generate an interrupt to the CPU before the start of the next frame. If the interrupt enable flag in the status word is set, the 8275 generates one interrupt IRQ at the beginning of the last display row. This interrupt can be used to reinitialize the DMA for refreshing, initializing the timers used for generating SYNC pulses.

Output signals which enhance the capability of the 8275 are as follows :

(1) Light Enable (LTEN)

(2) Video Suppression (VSP)

(3) Highlight (HLGT)

(4) Line Attributes (LA_0 and LA_1)

These are used to provide field attributes such as blink, underline, reverse video etc. etc. and character attributes which give limited graphics facility.

The 8275 uses 4 special codes also. They are listed below :

(1) End of row

(2) End of row, Stop DMA

(3) End of screen

(4) End of screen, Stop DMA

Their function is to blank the screen starting from the time when the special code is detected by the 8275 upto the end of the current row being scanned or screen, depending upon the code. Stop DMA option stops DMA requests for this duration.

The 8275 recognizes the displayable character codes or the attribute by checking the MSB. If MSB is HIGH, the character is recognized as an attribute otherwise it is treated as a displayable character and the code after suppressing this MSB (i.e. 7 bit code) is sent to the character generator ROM. In case of an attribute, an appropriate action is taken.

To display the cursor, 8275 maintains two registers which are loaded by the row number and the column number of the cursor. The cursor can be programmed to appear on the display as :

(1) A blinking underline
(2) A blinking reverse video block
(3) A non-blinking underline
(4) A non-blinking reverse video block.

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

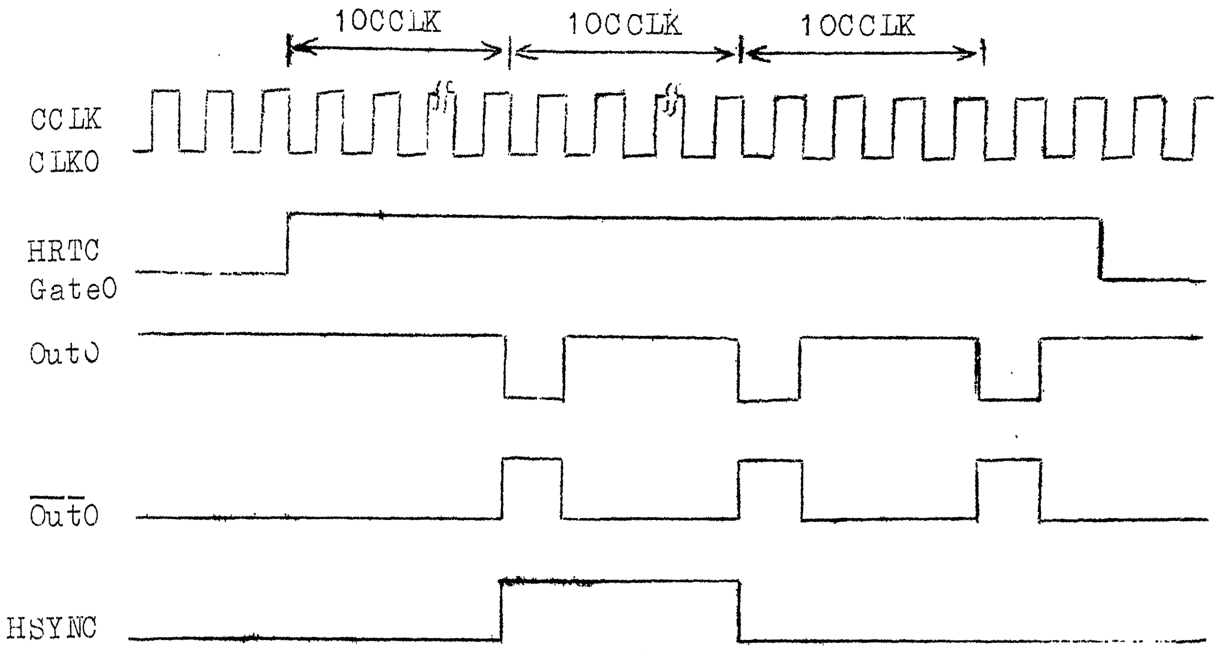
3.3 COMPOSITE VIDEO GENERATION

As already described 8275 CRT Controller gives two output signals HRTC and VRTC programmable by software. During these timings Video is disabled by activating video suppression signal. Horizontal and vertical synchronizing pulses are derived using INTEL Programmable interval timer 8253. As timings of these pulses can again be changed by software, this scheme is compatible to the facility provided by the 8275. Thus this design may be connected to any T.V. monitor. The timing requirements for that T.V. monitor can easily be obtained by appropriate software commands. These parameters are stored in

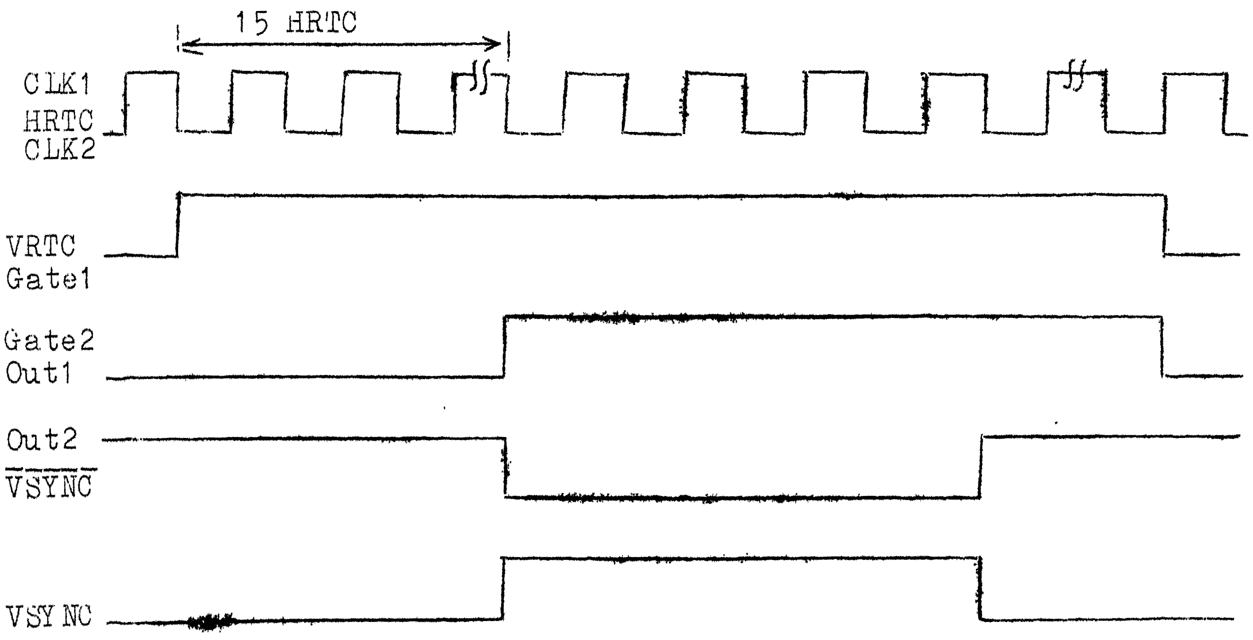
RAM and can be modified through keyboard.

8253 has three programmable timers. Out of these three, one is used to generate HSYNC and two for VSYNC. The timer ZERO used to generate HSYNC has been used in mode 2. (Rate generator or Divide by N mode); the output goes low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. The gate input, when low, forces the output high. When it goes high, the counter starts from the initial point. In this scheme HRTC has been connected as gate input and character clock as the clock input. The count loaded in count register is 10. This Timer output is fed to data-synchronizer which gives the HSYNC as shown in Fig. 3.6.

Timer 1, employed to generate VSYNC, has been used in Mode 0 and Timer 2 in Mode 1. In mode 0 the output is initially low after the mode set operation. After the count is loaded into the selected count register, the output remains low and the counter starts counting. When terminal count is reached, the output goes high and remains high until the select register is reloaded with the new count. In this scheme VRTC has been used as gate input and



Counter 0(8253) Waveforms to generate HSYNC



Counter 1&2 (8253) waveforms to generate VSYNC

Figure 3.4

Waveforms of 8253 Co . .

HRTC as clock. The count loaded is 15, so after 15 HRTC, the output of this timer goes high. This high transition is used to trigger the counter 2 operating in mode 1. In mode 1, (programmable one shot), the output goes low on the count following the rising edge of the gate input. The output goes high on the terminal count. In the present case the count loaded is 4, so counter 2 output remains high until the trigger is received from counter 1, then it goes low, remains low for 4 HRTC and finally comes back to high as shown in Fig. 3.6.

As already described, 8275 issue an interrupt request (IRQ) at the beginning of the last display row. At this time the CPU initializes these timers (which is before the first retrace row). Thus counter 2 output is obtained in each frame with its low duration for 4 HRTC. It is used as VSYNC.

The hardware realization to achieve serial video output from the character code supplied by 8275 is as shown in Fig. 3.5. 8275 gives the character code output on the falling edge of the character clock. Keeping in view the delays of the 8275, and those of the character generator ROM S6834, these codes are latched on the rising edge of CCLK. This has been done by using a 74174 HEX D FF. The output of 74174 is connected to the A_3 - A_7 pins of character generator ROM. Thus the input to this ROM is kept stable for the entire clock duration. A_0 - A_2 pins of the character generator ROM are connected to the latched line count outputs (LC0-2) of 8275. (Though 8275 gives a 7 bit character code, only 6 have been used here. CC6 has been ignored as 6 bits can give the required decoding of the character). The output of the character generator ROM is loaded into the parallel-serial shift register 74166 which gives the serial video out at dot clock frequency. To derive character clock, one 74163 synchronous 4 bit binary counter has been used which divides dot clock by 7 to give the character clock. The load input for 74163 has been derived from $\overline{QA.QB.QC}$ where QA, QB and QC are outputs of the 74163. To give the maximum allowance for character generator ROM delay, the same load input is not used for 74166 but instead, is derived from QA + QB. The character code output from S6834 is loaded into 74166 at the rising edge of the clock following the low value of the load input. Attributes like LTEN, VSP, HLGT are also latched using 74174s. As shown in Fig. 3.7 the 74166 starts giving the serial output only after 2 character clocks; these attributes are therefore latched twice. The composite video is derived by using open collector inverters, as shown in Fig. 3.5.

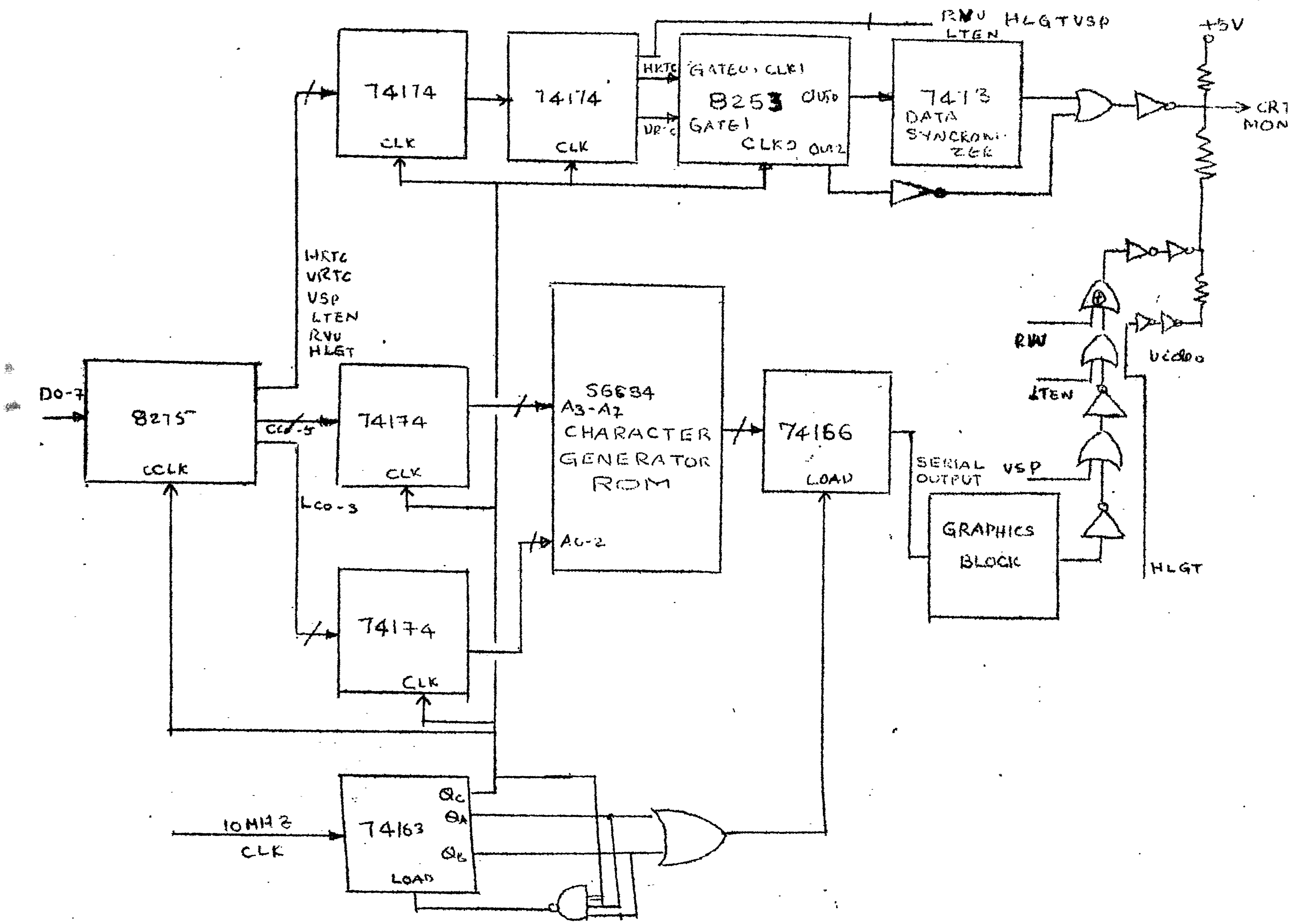


FIG 3.5 GENERATION OF COMPOSITE VIDEO

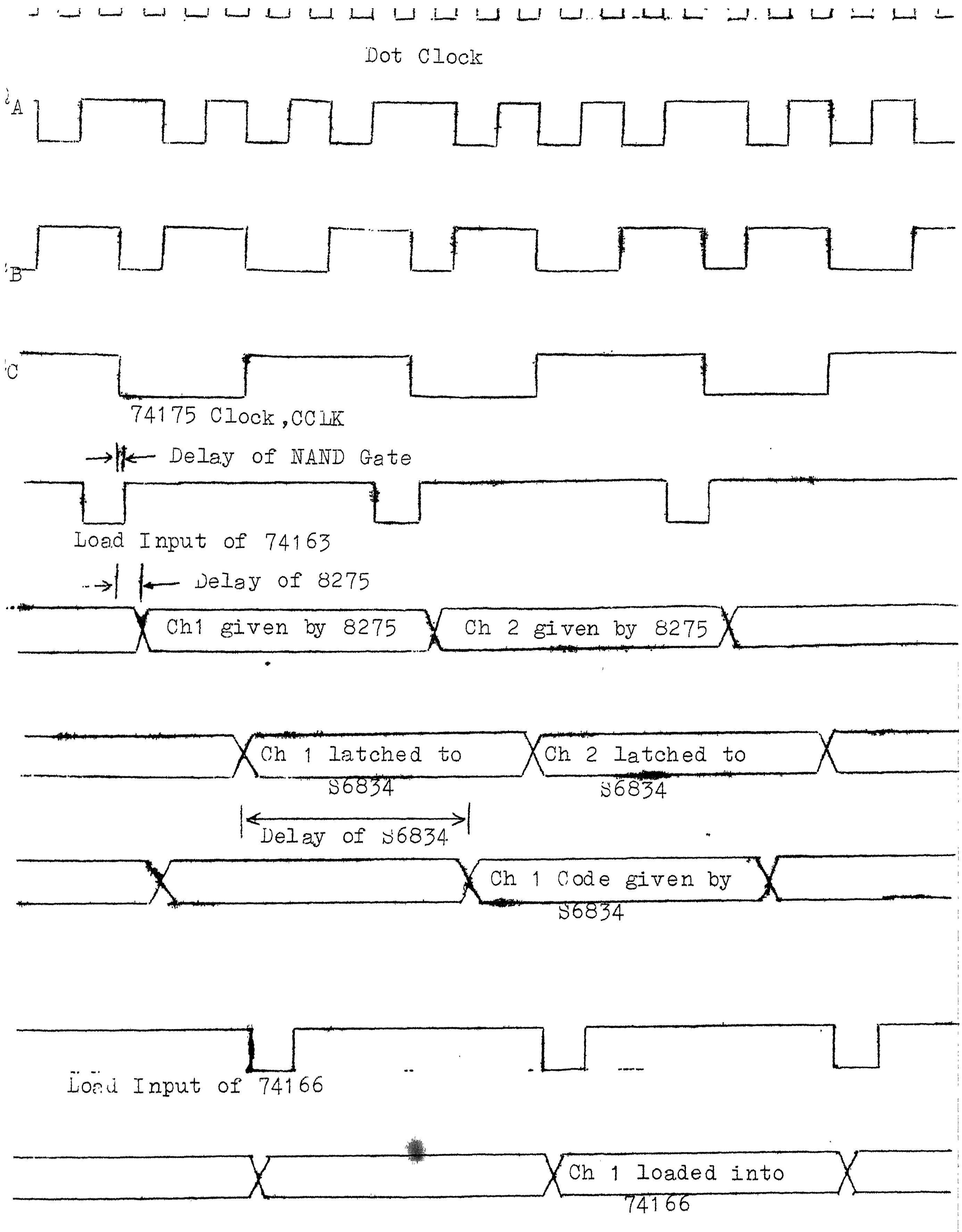


Fig. 3.6: Dot Timing Logic

Thus this provides 4 levels of composite video. SYNC level, the minimum one, and Highlight, the maximum one. The two other levels which lie in between are black and white levels.

3.3.1 <u>VIDEO SPECIFICATIONS</u> : Each character is a cell of 7 X 11 dots. Top and Bottom lines of each row are blanked. Actual character lies between 2nd & 8th line. 9th line is again blank. Cursor or underline placement will be in the 10th row.

Characters per row (m)	= 64
lines per row (q)	= 11
rows per frame (n)	= 24
HRTC duration (m')	= 32 Character Clocks
VRTC duration (n')	= 44 HRTC
HSYNC	= 10 dot clocks
VSYNC	= 4 HRTC
Front Porch in HRTC	= 10 dot clocks
Front Porch in VRTC	= 15 HRTC
character clock frequency	= 10 ÷ 7 MHZ

3.4 DMA CONTROLLER

Characters displayed on the CRT screen are to be refreshed at a fast rate to make the pattern stable. This calls for the need of fast data transfer from the display memory to the CRT Controller. If this task is assigned to the CPU, it would not be sufficiently fast as can be seen from the following considerations. The routine which would do so will have to have the following instructions :

(1) Fetch the address of the characters to be transferred.

(2) Fetch the required character from this address and send to the CRT Controller.

(3) Check whether it was the last character of the screen.

if yes, initialize the address

if no, increment the address.

This routine would easily take around 50-60 instructions i.e. around 80-100 microsecs. This is quite slow compared to our requirement in which roughly 60-80 characters are to be retrieved from memory in 0.7 - 0.8 ms maximum i.e. one character just in 10 microsecs. Hence the need arises of a device which could result in the fast data transfer. INTEL 8257 Programmable DMA Controller is the device which solves this problem. This chip can be programmed to supply all 60-80 characters in one burst which would take around 128 microsecs only. The complete schema of such a DMA based system, using a 8085 CPU is shown in Fig. 3.7.

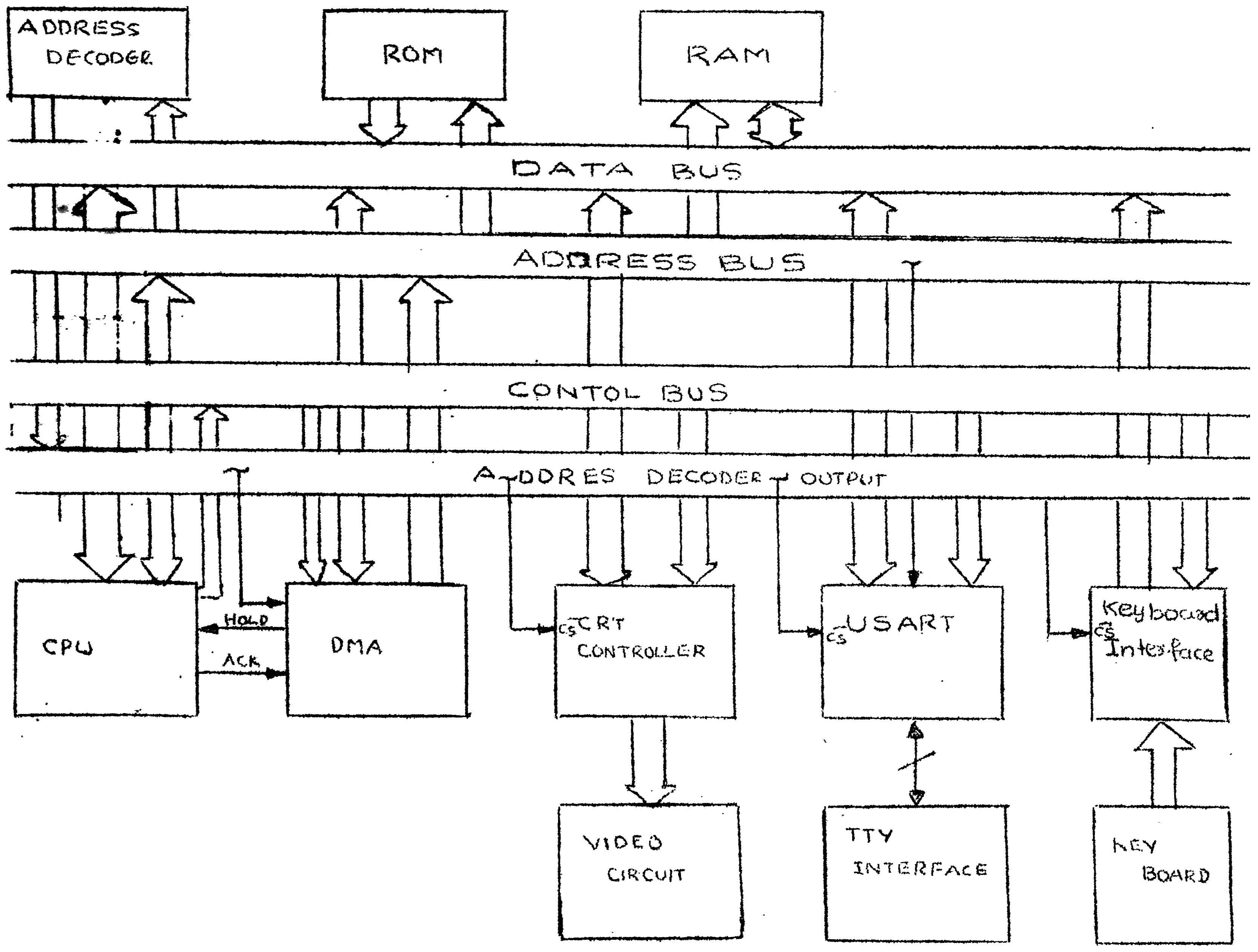


FIG 3.7: BASIC SCHEMATIC OF CRT TERMINAL

3.4.1 Programmable DMA Controller INTEL 8257

This is a 4 channel direct memory access Controller. Its primary function is to generate, upon a peripheral request, a sequential memory address which will allow the peripheral to read or write data directly from or to memory. During system initialization, DMA channel 2 address register is initialized with the first displayable memory location and channel 2 terminal count register is loaded with (i) the no.of characters per screen (ii) Mode of DMA. This chip has been used in the Auto load feature which duplicates channel 2 registers contents into channel 3 registers during initialization. Whenever 8275 requests DMA for data transfer by pulling its DRQ (DMA request line) high, DMA issues an Hold request (HRQ) to CPU. CPU acknowledges this by giving Hold Acknowledge (HLDA). Upon getting HLDA signal, DMA latches the upper byte of address on the higher byte of address through its data byte and lower byte directly on lower byte of address. It then issues MEMORY READ and DMA acknowledge signals. In the next cycle it issues I/O write signal to 8275 which takes the data lying on data bus (put by the memory) and puts it into its row buffer. DMA then increments the address register and decrements the count. Whenever this count goes to zero (it would happen when the last character of the frame has been sent) channel 3 contents are duplicated nondestructively into channel 2 registers. Thus Auto load feature saves CPU overhead of reinitializing the DMA address registers and count registers at the end of every frame.

3.4.2 Bus Sharing between DMA & CPU

INTEL 8085 microprocessor and 8257 DMA both use multiplexed address-data bus. In 8085 lower address byte is muliplexed with data bus while in 8257 the higher address byte is multiplexed with data bus.

In the first clock cycle of a machine cycle (one machine cycle is one complete memory access i.e. either read or write which consists of 3-5 clock cycles) 8085 issues an ALE signal (Address latch enable). This time AD_{0-7} lines of processor carry lower 8 bits of address. In the next clock cycle this bus is floated i.e. address gets lost and in the 3rd clock cycle data coming from memory appears on data bus. To get the proper address the falling edge of ALE has been used to latch this address.

When DMA is the bus-master, AEN is high which puts the output of this 8282 in high impedence state. As address bus of CPU and DMA are common, this feature is essential so that when one device is driving the bus, the other is not interfering.

Similarly when DMA is driving the bus (CPU in the Hold state), it presents higher bits of address (address stored in the address register of DMA) on the data bus. This time it issues one ADSTB (Address Strobe) signal which is used to latch this address in the same way as ALE is used for 8085. In this case $\overline{\text{AEN}}$ is used as $\overline{\text{OE}}$ (output enable) of 8282.

3.5 Bus & I/O Control

The data bus of CPU, Memory section and I/O section is buffered using 8833 bidirectional buffers. This is essential to avoid loading of any of these lines. In this scheme 2 sets of 8833's have been used. One set interfacing between memory and the CPU and the other one interfacing between I/O devices and CPU. The set which interfaces between memory and CPU uses $\overline{\text{MEMR}}$ as driver enable input and $\overline{\text{MEMW}}$ as receiver enable input. Thus when any read operation is taking place, receiver goes to high impedence state and data available on driver input appears on bus. Similarly, whenever any write is being performed, driver goes to high impedance state and data on bus pins appear on receiver pins from where it is taken by the selected device.

8085 CPU provides $\overline{RD}$, $\overline{WR}$ and $IO/\overline{M}$ output signals which are low level active. DMA provides $\overline{MEMR}$, $\overline{MEMW}$ as output and $\overline{IOR}$ & $\overline{IOW}$ as bidirectional signals. To make these signals compatible 8085 lines have been decoded to give $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$ and $\overline{IOW}$. It has been accomplished by using the tristate quad 2 to 1 multiplexer 74S257 in which HLDA is used as output control signal. Thus when CPU is in HOLD state, these lines are free for use by DMA. $IO/\overline{M}$ is used as select input.

As 74S257 is a Schottky device while the outputs of 8085 can only drive a single normal TTL load, these CPU output lines i.e. RD, WR, $IO/\overline{M}$, HLDA are buffered using a 74365 hex buffer.

3.6 I/O Decoding

There are two distinct ways for handling input output (I/O) in any microprocessor-based system : (i) I/O mapped I/O, and (ii) Memory mapped I/O. Memory Mapped I/O means that I/O devices are also treated as memory devices, thus enabling the user to exploit the more versatile instructions provided for memory handling. This, however, results in the reduction of actual physical memory accessible by the system. In the present design as the memory requirement is sufficiently high (because these CRT terminals have to

work as independent work stations) an I/O mapped I/O scheme has been chosen. Each I/O device is assigned an 8 bit address which appears both as the lower byte and as the upper byte of the address, given out by the CPU in response to an I/O access instruction (IN or OUT). As a sophisticated I/O device has at least 2-3 internal registers, linear select scheme can at most permit the selection of 2-3 I/O devices. The present design incorporates 5-6 LSI's, with the total number of addressable registers exceeding twenty. The select signals have therefore been derived by decoding the address lines. As the DMA Controller is also used in I/O mapped I/O scheme, one is not allowed to decode the lower 4 bits which form actual address bits connected to the DMA Controller chip. Hence we are just left with the 4 higher bits $A_4 - A_7$. As the present design uses only 8 I/O devices, the decoding scheme has utilized only three of these bits, viz. $A_5 - A_7$. A 3 line to 8-line decoder (74155) has been used for this purpose. When DMA is the bus master, this selection should be inhibited to avoid any wrong selection, for this strobe of 74155 is derived from AEN + $\overline{\text{IOR}}$. $\overline{\text{IOW}}$.
The various addresses of various I/O devices' registers are given in Table 3.1.

3.7 Keyboard interface

In the present design, keyboard servicing is

Table 3.1 <u>I/O Addressing Scheme</u>

Address Lines									
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	I/O Device	Register
0	0	0	0	0	0	0	0	8255	Port A
0	0	0	0	0	0	0	1		Port B
0	0	0	0	0	0	1	0		Port C
0	0	0	0	0	0	1	1		Control
0	0	1	0	0	0	0	0	8291	Data In/Data Out
0	0	1	0	0	0	0	1		Int.Status 1/Int. Mask 1
0	0	1	0	0	0	1	0		Int.Status 2/Int. Mask 2
0	0	1	0	0	0	1	1		Serial Poll Status/ Serial Poll Mode
0	0	1	0	0	1	0	0		Addr.Status/Addr. Mode
0	0	1	0	0	1	0	1		Command/Aux.Mode/ Addr.Mode
0	0	1	0	0	1	1	0		Addr.0/Addr.0/1
0	0	1	0	0	1	1	1		Addr.1/EOS
0	1	0	0	0	0	0	0	8257	Ch0 address register
0	1	0	0	0	0	0	1		Ch0 terminal count register
0	1	0	0	0	0	1	0		Ch1 address register
0	1	0	0	0	0	1	1		Ch1 terminal count register
0	1	0	0	0	1	0	0		Ch2 address register
0	1	0	0	0	1	0	1		Ch2 terminal count register
0	1	0	0	0	1	1	0		Ch3 address register
0	1	0	0	0	1	1	1		Ch3 terminal count register
0	1	0	0	1	0	0	0		mode word
0	1	1	0	0	0	0	0	74126	Keyboard Data Buffer
1	0	0	0	0	0	0	0	8253	Counter 0 Count
1	0	0	0	0	0	0	1		Counter 1 Count
1	0	0	0	0	0	1	0		Counter 2 Count
1	0	0	0	0	0	1	1		Mode register
1	0	1	0	0	0	0	0	8275	Parameter
1	0	1	0	0	0	0	1		Command/Status
1	1	0	0	0	0	0	0	8251	Data
1	1	0	0	0	0	0	1		Command
1	1	1	0	0	0	0	0	74C173	Keyboard STB FF

done by polling and not by interrupt. At the end of each frame, the CPU reads the status of an FF which tells whether some character has been received from keyboard or not. A quad D-FF 74C173 is used for this purpose, with the D-input HIGH and STB signal of keyboard used as clock. Thus whenever a key is pressed, the STB comes and sets Q_K. CPU periodically reads the status of Q_K on its Do line. If $D_0 = 1$ the CPU takes data from the keyboard data buffer and resets Q_K.

3.8 USART Interface

A USART has been provided for serial communication with TTY or some similar work-station. It has been used in the asynchronous mode as the speed requirement is not very high. Provision has been made to give different clock inputs to USART depending upon the need. The clock to interface with TTY (110 band) has been derived by using a 555 timer used in ASTABLE mode.

The USART generates two output signals TxRDY and RxRDY. TxRDY goes high when all the bits of the character supplied by CPU has been serially transmitted. RxRDY goes high when USART receiver receives a character. In certain applications the need may arise to generate two separate interrupts; one by

RxRDY and other by TxRDY. In the present design provision has been made for this facility but at present these two lines have been ORed to generate a single interrupt and it has been left to CPU to determine the source of the interrupt. The CPU does so by reading the status word of 8251. This single interrupt is connected through a switch to the RST 7.5 pin of 8085.

3.9 Memory Organization

Any microprocessor-based CRT terminal requires a certain amount of ROM space to store the system software and some RAM area to store the user's text/programs. In the present design the RAM space has been chosen to be 8 K bytes which would permit the display of 5 pages. As already pointed out, the programs to be run here would not be too lengthy and so this much space should be sufficient. The system software listed in Appendix suggests that at least 4 K bytes of ROM should be provided. In this design 8 K has been provided to facilitate future expansion as well as execution of user's program stored in such a ROM. The actual chips which have been used are INTEL 2114 1024 x 4 - bit RAMs and INTEL 2716 2048 x 8 - bit EPROMs.

As the sizes of the ROMs and RAMs are different, simple decoding of the address lines would not serve the purpose. To be more specific if $A_{11} - A_{14}$ are decoded to select the different devices it would result in non-Contiguous RAM area as the RAM uses the bits $A_0 - A_9$ only for its addressing.

The actual allocation of the address bits for selecting the various memory devices so that the areas are contiguous is given in Table 3.2, which also indicates the serial number of these devices expressed in terms of 4 bits DCBA. Thus if these bits are generated from the address bits by a suitable Combinatorial circuit, one can simply use a 4 bit to 16-line decoder for obtaining the necessary control outputs.

The Karnaugh Map for DCBA in terms of A_{13}, A_{12}, A_{11} and A_{10} is given in Fig.3.8 which also gives the required Boolean expressions. To realize these expressions one quad 2 to 1 multiplexer 74157 is used with its strobe input kept always low (enable). The strobe inputs $\overline{G_1}$ and $\overline{G_2}$ of 74154 are generated as follows :

$$\overline{G_1} = A_{15}$$
$$G_2 = \overline{MEMR} \cdot \overline{MEMW}$$

Table - 3.2 Memory Allocation Scheme

	Address Lines									
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	D	C	B	A
ROM-1	0	0	0	0	0	X	0	0	0	0
ROM-2	0	0	0	0	1	X	0	0	0	1
ROM-3	0	0	0	1	0	X	0	0	1	0
ROM-4	0	0	0	1	1	X	0	0	1	1
RAM-1	0	0	1	0	0	0	0	1	0	0
RAM-2	0	0	1	0	0	1	0	1	0	1
RAM-3	0	0	1	0	1	0	0	1	1	0
RAM-4	0	0	1	0	1	1	0	1	1	1
RAM-5	0	0	1	1	0	0	1	0	0	0
RAM-6	0	0	1	1	0	1	1	0	0	1
RAM-7	0	0	1	1	1	0	1	0	1	0
RAM-8	0	0	1	1	1	1	1	0	1	1

$A_{13}A_{12}$ \ $A_{11}A_{10}$	00	01	11	10
00	0000	0000	0001	0001
01	0010	0010	0011	0011
11	1000	1001	1011	1010
10	0100	0101	0111	0110

$$A = \overline{A_{13}} \cdot A_{11} + A_{13} \cdot A_{10}$$

$$B = \overline{A_{13}} \cdot A_{12} + A_{13} \cdot A_{11}$$

$$C = A_{13} \cdot \overline{A_{12}}$$

$$D = A_{13} \cdot A_{12}$$

Fig. 3.8
Karnaugh Map Realization

CHAPTER - 4

SYSTEM SOFTWARE

The total software can be broadly divided into 3 main parts - (i) Main Routine, (ii) RST 7.5 Interrupt Routine and (iii) TRAP Routine. The main routine is initiated when the system is reset. At this time all interrupts other than TRAP are automatically disabled. The TRAP interrupt, connected to the IRQ output of 8275, also remains indirectly disabled while the main routine is being executed until a Start Display Command is given to 8275, thereby enabling its IRQ. This command is given after the main routine completes the initialization of the various registers of the I/O devices as also some of its own memory locations which are referred during the execution of other routines. The complete flow chart of the main routine is given in Fig.4.1 which is self explanatory.

The CPU idles at the last instruction of this routine until it receives an interrupt. An interrupt cases the CPU to jump either to the TRAP routine or to the RST 7.5 routine depending on the interrupt. All other routines are called and executed within one of these two interrupt routines. After servicing the interrupt the CPU always returns to the idle state at the last instruction of the main routine.

Main Routine

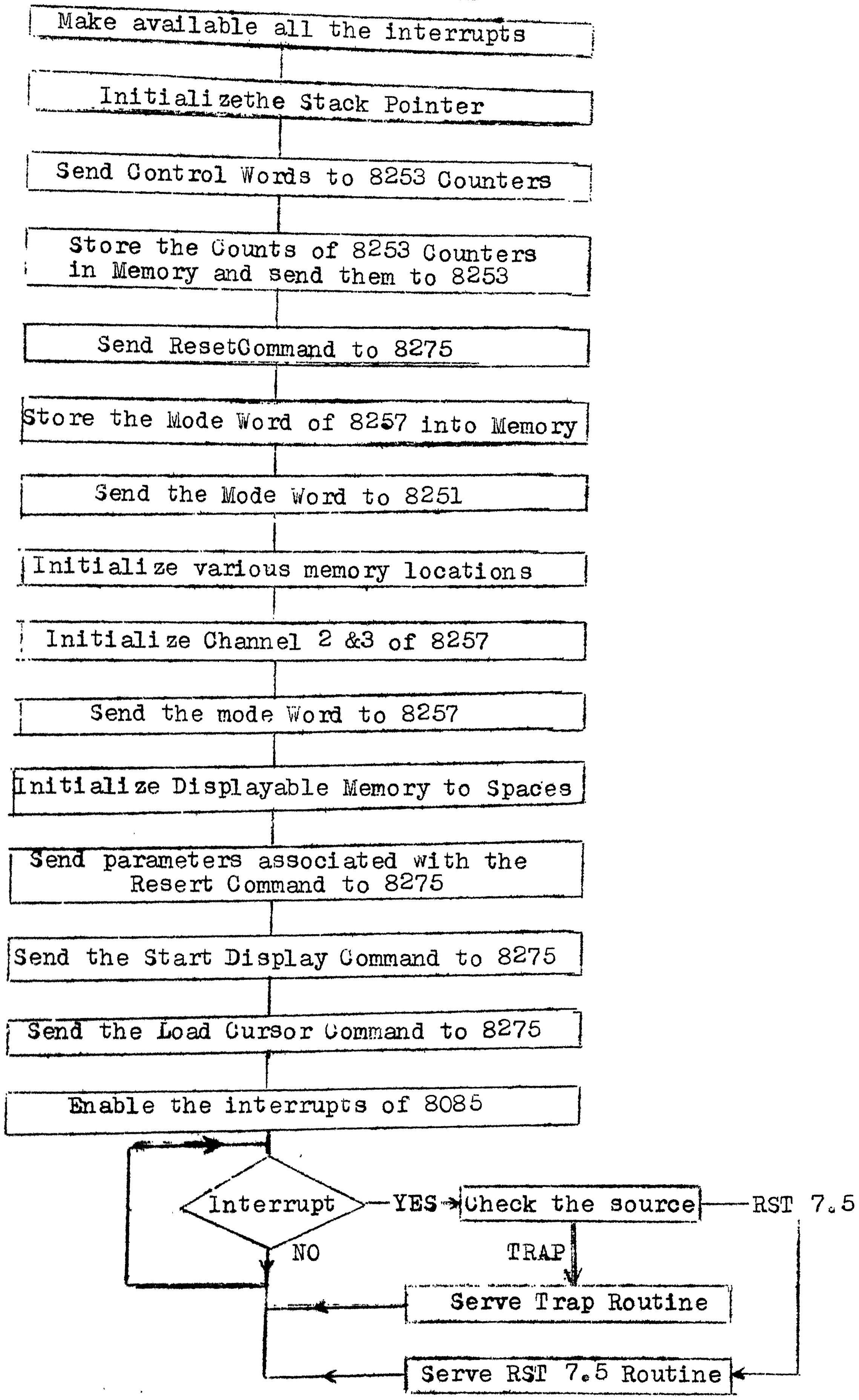

Make available all the interrupts
Initializethe Stack Pointer
Send Control Words to 8253 Counters
Store the Counts of 8253 Counters in Memory and send them to 8253
Send ResetCommand to 8275
Store the Mode Word of 8257 into Memory
Send the Mode Word to 8251
Initialize various memory locations
Initialize Channel 2 &3 of 8257
Send the mode Word to 8257
Initialize Displayable Memory to Spaces
Send parameters associated with the Resert Command to 8275
Send the Start Display Command to 8275
Send the Load Cursor Command to 8275
Enable the interrupts of 8085
Interrupt
YES
NO
Check the source
RST 7.5
TRAP
Serve Trap Routine
Serve RST 7.5 Routine

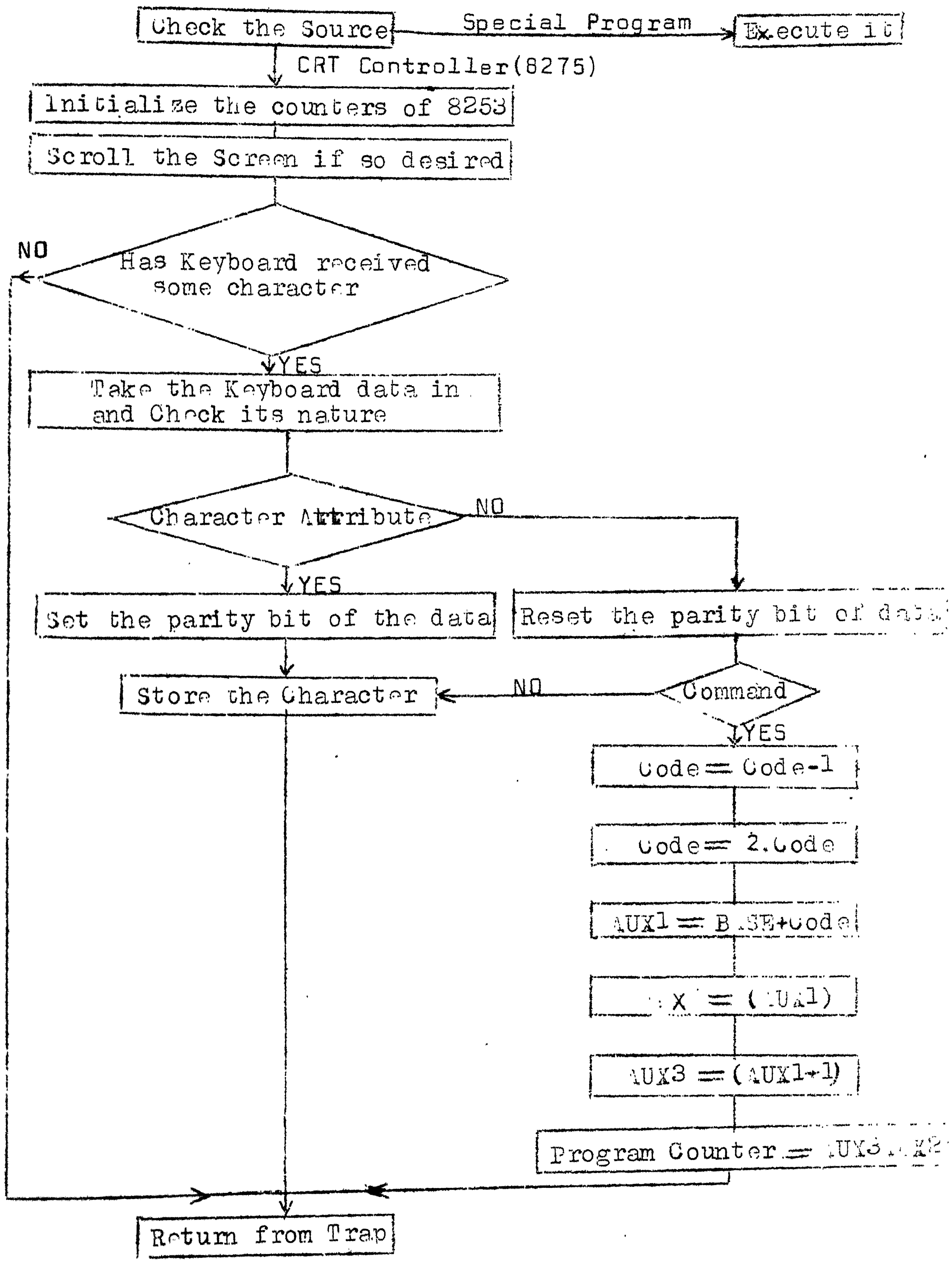
Fig. 4.1

Among the interrupts used in this system, TRAP is unmaskable and has the highest priority, while RST 7.5 is a positive logic edge sensitive interrupt and is next to TRAP in priority. TRAP can come either from the keyboard or from some special programs. The detailed flow chart is given in Fig. 4.2.

The execution of various commands within the TRAP routine is done as follows : Each command is associated with a key which generates a code on pressing the key. Command Codes are less than 20 Hex. So if any key, whose code is less than 20 Hex (after resetting the MSB), is pressed, CPU recognizes it as a command and calls a routine. To find the address of the Command routine, this routine decrements the Command Code by 1 and then doubles this figure. Now it is added to a fixed address BASE. The resulting sum points to the lower byte of the address of the Command routine, with the byte of the higher address placed in the next location. These two locations are accessed and then the program counter is initialized to this value which results in the execution of the desired routine.

RST 7.5 interrupt can be caused either by USART or by the keyboard. In the beginning of this routine the source is checked and USART routines are

TRAP



Check the Source
Special Program
Execute it
CRT Controller(8275)
Initialize the counters of 8253
Scroll the Screen if so desired
Has Keyboard received some character
NO
YES
Take the Keyboard data in and Check its nature
Character Attribute
NO
YES
Set the parity bit of the data
Reset the parity bit of data
Store the Character
NO
Command
YES
Code = Code-1
Code = 2.Code
AUX1 = BASE+Code
AUX2 = (AUX1)
AUX3 = (AUX1+1)
Program Counter = AUX3.AUX2
Return from Trap

FIG. 4.2

executed if the source is the USART; otherwise it is assumed that the interrupt has come from the keyboard and some parameters are being changed by the user. These routines are discussed in details in the coming sections.

The various routines used in the system fall under five broad functional categories :

(1) Control of inter-station communication
(2) Control of the cursor location
(3) Storage of the keyed-in character
(4) Execution of editing commands
(5) Special functions

These routines are discussed in sections 4.2. - 4.6 under the functional heads, after explaining the various parameters necessary for the design of the system software in the next section.

4.1 DESCRIPTION OF PARAMETERS

There are two sets of parameters used in developing the system software. One set consists of the Commands and parameters to be sent to the I/O devices for their proper functioning and the other set consists of various parameters which are used by the CPU itself to handle the various routines. Each of these parameters has been assigned a specific location in the first page of the RAM area (upper byte of address is 20 Hex) and the value of any of these parameters may be changed under the control of the Parameter Storage Routine (discussed in section 4.6) by specifying the lower byte of the address and the value of the parameter. This provides the user the facility of programming associated peripherals according to his needs. Table 4.1 lists all such parameters alongwith their locations in the memory and explanation of their functions.

4.2 CONTROL OF INTER-STATION COMMUNICATION

As has already been pointed out, inter-station communication is achieved through USART. The routines which fall in this catagory are the following :

(1) Start Transmission
(2) Send the next character to USART
(3) Stop Transmission
(4) Start Reception
(5) Receive the next character from USART
(6) Stop Reception

Routines 1,3,4 and 6 are invoked by pressing some specific keys on the keyboard and executed under TRAP while the routines 2 and 5 are invoked by the

Table 4.1 : Software Parameters

PARAMETER	ADDRESS IN HEX	EXPLANATION
HSYNC	2001	HSYNC pulse; Count, to be loaded into counter 0 of 8253.
VBLFP	2002	VERTICAL BLANKING FRONT PORCH; Count to be loaded into counter 1 of 8253.
VSYNC	2003	VSYNC Pulse; Count to be loaded into counter 2 of 8253.
M8257	2004	MODE WORD OF 8257 (DMA)
CH0A	2005 2006	DMA CHANNEL 0 ADDRESS
CH0TC	2007 2008	DMA CHANNEL 0 TERMINAL COUNT WORD
CH1A	2009 200A	DMA CHANNEL 1 ADDRESS
CH1TC	200B 200C	DMA CHANNEL 1 TERMINAL COUNT WORD
TOPLP	200D 200E	TOP LEFT POINTER - Absolute Address of the first character to be displayed on CRT screen
CH2TC	200F 2010	DMA CHANNEL 2 TERMINAL COUNT WORD
M8251	2011	MODE WORD OF 8251 (USART)
CSTTX	2012	COMMAND TO START TRANSMISSION
CSPTX	2013	COMMAND TO STOP TRANSMISSION
CSTRX	2014	COMMAND TO START RECEPTION
CSPRX	2015	COMMAND TO STOP RECEPTION
PARA 1	2016	PARAMETER 1 OF RESET COMMAND TO 8275
PARA 2	2017	PARAMETER 2 OF RESET COMMAND TO 8275
PARA 3	2018	PARAMETER 3 OF RESET COMMAND TO 8275
PARA 4	2019	PARAMETER 4 OF RESET COMMAND TO 8275
STDC	201A	START DISPLAY COMMAND TO 8275
SACLM	201B 201C	START ADDRESS FOR CLEAR MEMORY ROUTINE
LACLM	201D 201E	LAST ADDRESS FOR CLEAR MEMORY ROUTINE
DSASM	201F 2020	DUMP START ADDRESS OF SOURCE MEMORY FOR DUMP ROUTINE
DCASM	2021 2022	DUMP LAST ADDRESS OF SOURCE MEMORY FOR DUMP
DSADM	2023 2024	DUMP START ADDRESS OF DESTINATION MEMORY FOR DUMP ROUTINE
HICHA	2025,2026	HIGHEST ALLOWED CHARACTER ADDRESS
M8255	2027	MODE WORD OF 8255
BSR55	2028	BIT SET RESET WORD FOR 8255
TxSTA	2029 202A	TRANSMISSION START ADDRESS - Gives the address from where the first character is to be transmitted when START TX Key is pressed.
RxSTA	202B 202C	RECEPTION START ADDRESS - Gives the address where the first character received from USART is to be stored.

Contd....

PARAMETER	ADDRESS IN HEX	EXPLANATION
SKPCT	202D	SKIP COUNT - Gives the number of rows by which the screen is to be scrolled up when the SKIP Routine is invoked.
SCRUF	202E	SCROLL UP FLAG - Set whenever scroll-up condition is established; checked at the beginning of the TRAP routine for deciding the appropriate actions; reset after this checking, if set.
ESCF	202F	ESCAPE FLAG - Set if the ESCAPE Key is pressed, signifying that the succeeding character is to be interpreted as a character attribute to generate graphics; reset after getting this succeeding character.
PROTF	2030	PROTECTION FLAG - If set, tells that some protection zone exists.
INDEF	2031	INSERT OR DELETE FLAG - Set when any of the insert or delete routine is invoked; used in the Protection Check Routine which in turn modifies another flag NOIDF to indicate the further action to be taken, reset before the return from insert/ delete routine is executed.
NOIDF	2032	NO INSERT OR DELETE FLAG - If set, tells that an attempt has been made to fiddle with the protected zone and hence gives an indication that no action is to be taken.
PARAF	2033	PARAMETER FLAG - Set, when the first parameter to be changed is received through keyboard, used to send Reset Commands to 8275 and 8251; reset after PERIOD KEY is pressed in this mode.
CSRRW	2034	CURSOR ROW NUMBER
CSRCL	2035	CURSOR COLUMN NUMBER
CHCNT	2036	CHARACTER COUNT PER ROW
RWCNT	2037	ROW COUNT PER FRAME
INVAS	2038	INVALID ASCII CODE - Set, when an ASCII Code other than 30 Hex - 39 Hex OR 41 Hex - 46 Hex is detected. Further Execution of ASCII to Hex Conversion routine is stopped. Used, while execution of this converted code is initiated. Reset in the very beginning of ASCII to Hex Conversion Routine.

Contd...

PARAMETER	ADDRESS IN HEX	EXPLANATION
RWPTR	2041 2042	ROW POINTER - Represents the relative spacing of the first character in the cursor-row with respect to the first character on the screen. Thus if the cursor lies in the first row itself, the Row Pointer will be zero, if cursor lies in 2nd row, Row Pointer will be equal to CHCNT. Similarly if cursor lies in 3rd row, Row Pointer will be equal to 2XCHCNT and so on. This facilitates in calculating the absolute address of the first character in any row as follows : Absolute Address of the first character in a row = TOPLP + RWPTR
CSRA	2043 2044	CURSOR ABSOLUTE ADDRESS
LACHA	2045 2046	LAST CHARACTER ADDRESS - Tells the highest address which has been accessed by the program being entered.
CUTXA	2047 2048	CURRENT TRANSMISSION ADDRESS - Points to the address whose content is to be sent to USART upon interrupt from USART transmitter.
CURXA	2049 204A	CURRENT RECEPTION ADDRESS - Points to the address where the character received from USART is stored.
LOCHA	204B,204C	LOWEST ALLOWED CHARACTER ADDRESS
AFCPR	204F 2050	ADDRESS OF FIRST CHARACTER IN PROTECTED REGION
ALCPR	2051 2052	ADDRESS OF LAST CHARACTER IN PROTECTED REGION
TEMP1	2053 2054	TEMPORARY LOCATION 1 - Used in Insert/Delete Routine
TEMP2	2055 2056	TEMPORARY LOCATION 2 - Used in Insert/Delete Routine.
HEXSA	2057 2058	HEX CODE STARTING ADDRESS
ASCFA	2059 205A	ASCII CODE FIRST ADDRESS - Points to be first ASCII data to be converted to Hex.
ASCLA	205B 205C	ASCII CODE LAST ADDRESS - Points to the last data to be converted to Hex.
ACHFL	205D	ASCII CONVERSION TO HEX FLAG

Contd...

PARAMETER	ADDRESS IN HEX	EXPLANATION
		Used in ASCII to HEX Conversion Routine. Set, when the ASCII Character to be converted and placed in higher order 4 bits in fetched; reset, when the ASCII character to be converted and placed in lower order 4 bits is fetched.
TEMP3	205E	This location temporarily contains the higher order 4 bits of Hex Code. It is ORed with lower order 4 bits of Hex code after fetching this second ASCII character.
ASCSA	205F 2060	<u>ASC</u>II CODE <u>S</u>TARTING <u>A</u>DDRESS
HEXFA	2061 2062	<u>HEX</u> CODE <u>F</u>IRST <u>A</u>DDRESS - Points to the first hex data to be converted to ASCII.
HEXLA	2063 2064	<u>HEX</u> CODE <u>L</u>AST <u>A</u>DDRESS - Points to the last hex data to be converted to ASCII.
PROVF	2065	<u>PRO</u>TECTION <u>V</u>IOLATION <u>F</u>LAG - Set, when an attempt is made to store a character in a protected region, reset of otherwise; set by Protection check routine and used by Store Character routine.

USART itself by sending an RST 7.5 interrupt. Routines 1,3,4 & 6 are quite simple. While Routines 1 and 4 send one Start Transmission/reception command to USART as well as initialize the memory location to be addressed next, Routines 3 & 6 simply send Stop Transmission/receiption commands to USART. Flow charts for Routines 2 and 5 are given in Fig. 4.3.

4.3 Cursor Control Routines

These routines can be listed as follows :

(1) Cursor right

(2) Cursor left

(3) Cursor up

(4) Cursor home

(5) Line feed

(6) Carriage return

(7) Forward Tab

(8) Scroll up

(9) Scroll down

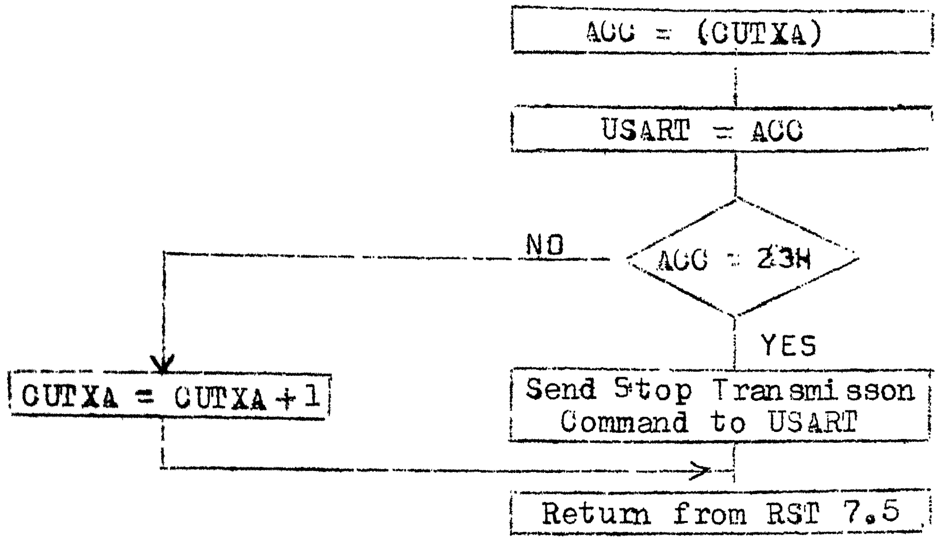
(10) Skip

These routines in turn call the following routines.

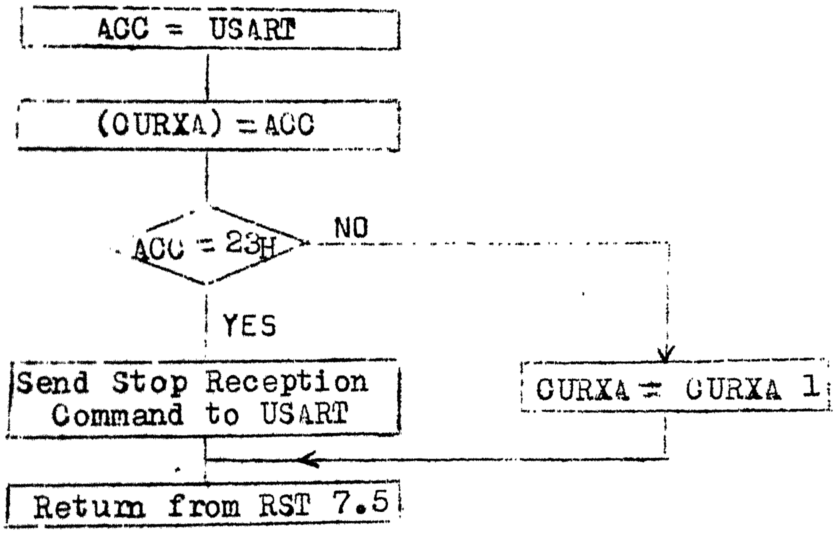
(1) Load Cursor

(2) DMA load

Flow Chart for sending a character upon Interrupt from USART



Flow Chart for receiving a character upon Interrupt from USART



NOTE: 23H (ASCII) denotes # sign. Thus is used as a delimiter of the text to be transmitted or received.

Fig. 4.3

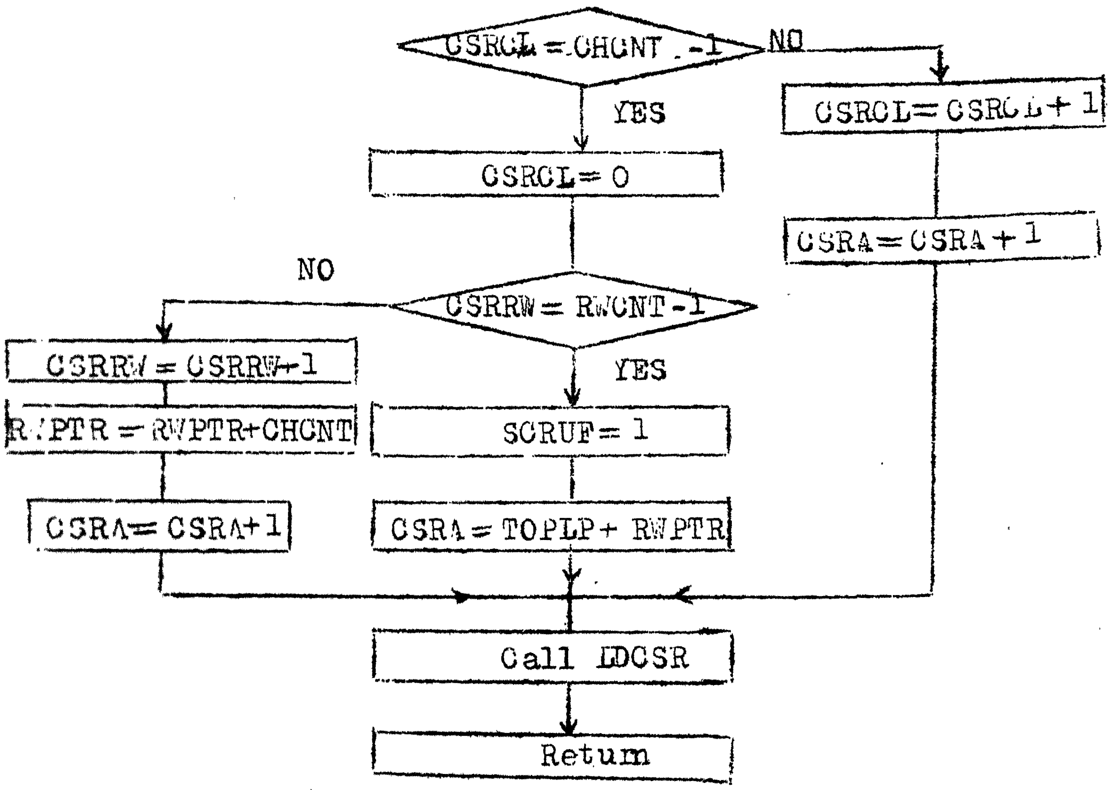
Load cursor routine sends the Load Cursor Command and its two parameters to 8275. These two parameters are stored in location CSRCL and CSRRW. DMA load routine initializes the address registers of channel 2 and channel 3 of DMA Controller to affect the display.

Cursor right routine advances the cursor by one column if cursor, before giving this command, lies in a column other than the last column of a row. If its present column is last one but row is other than last one, it is moved to the first column of the next row. In case cursor lies in the last row as well as in the last column, scroll condition is established and cursor is moved to the first character position. On the other hand Cursor left routine moves the cursor to the left by one column if it lies in a column other than the first column. The flow charts of these two routines are shown in Fig. 4.4.

Cursor up routine moves the cursor up by one row if its present row is other than the topmost row while Cursor home routine brings the cursor to the top left position (I Column of the I row) irrespective of its present location. Flow charts of these two routines are given in Fig. 4.5.

Line feed routine moves the cursor one row down (without affecting the column number) if it lies

Cursor right



Cursor left

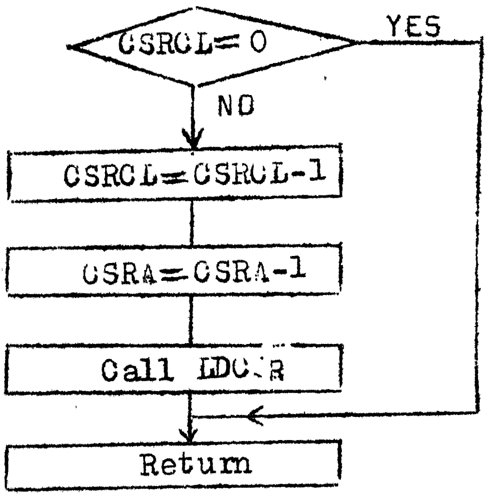
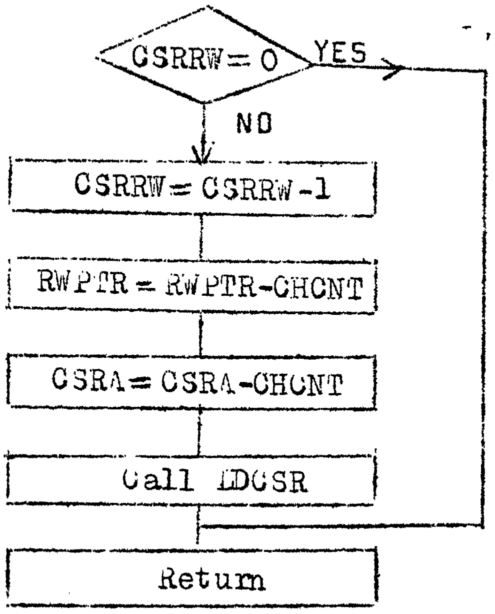


Fig. 4.4

Cursor Up



Cursor Home

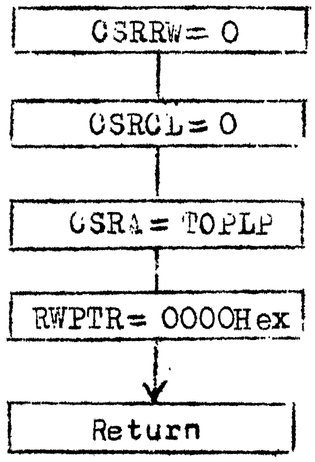


Fig. 4.5

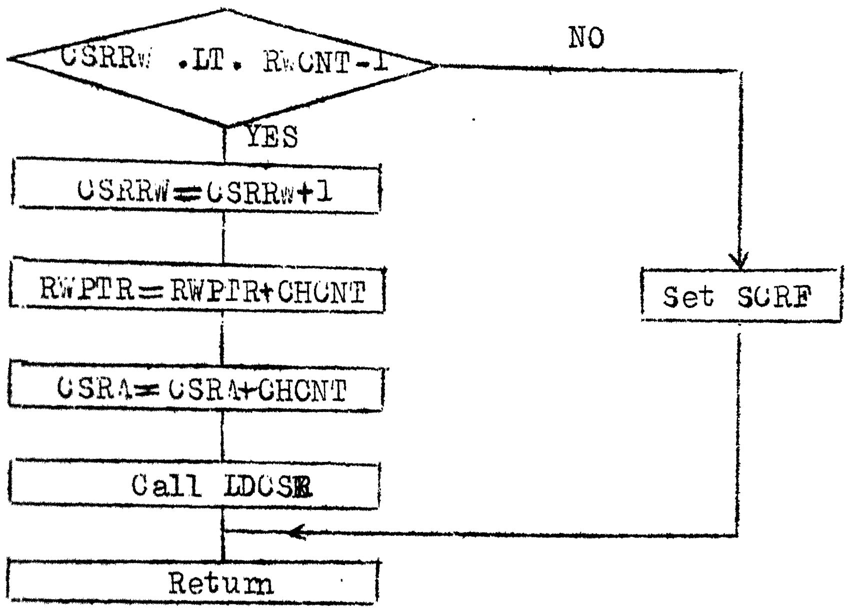
in a row other than the last row. In case of the last row, a scroll up condition is established. Carriage return routine does the same thing as Line feed routine does except that it brings the cursor to the I column of the next row. The flow charts are shown in Fig. 4.6.

Scroll routines affect the display on the screen. As the screen size is limited, these routines help the user to see his entire program in blocks while forward Tab routine modifies the cursor column number as given below:

Cursor Column number	
Before invoking this routine	After invoking this routine
0 - 7	7
8 - 15	15
16 - 23	23
24 - 31	31
32 - 39	39
40 - 47	47
48 - 55	55
56 - 63	63

Flow chart of these routines are given in Fig. 4.7.

Line Feed



Carriage Return

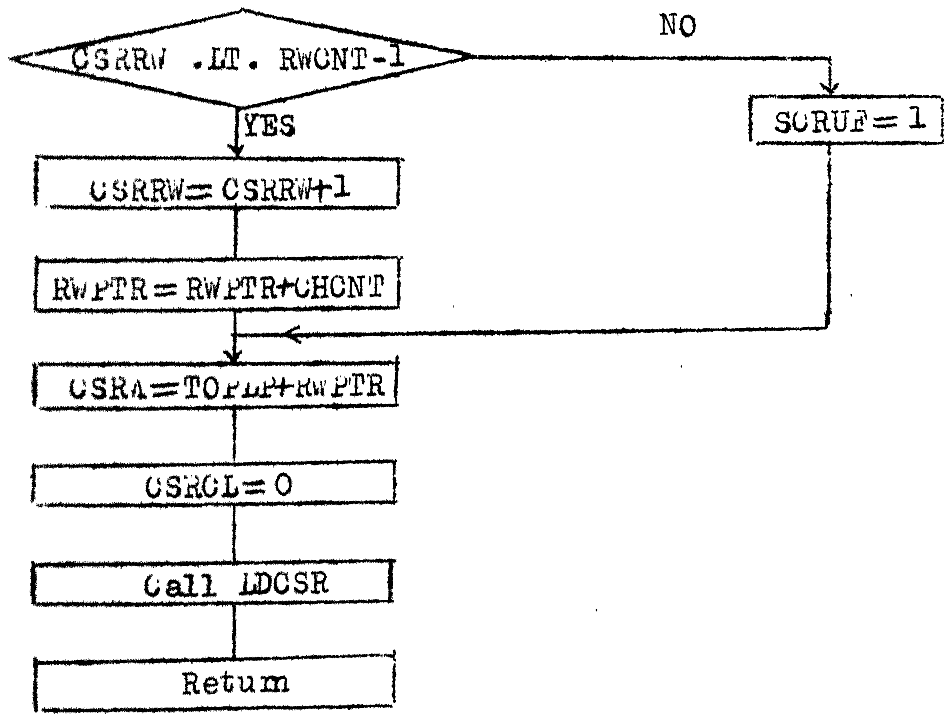


Fig. 4.6

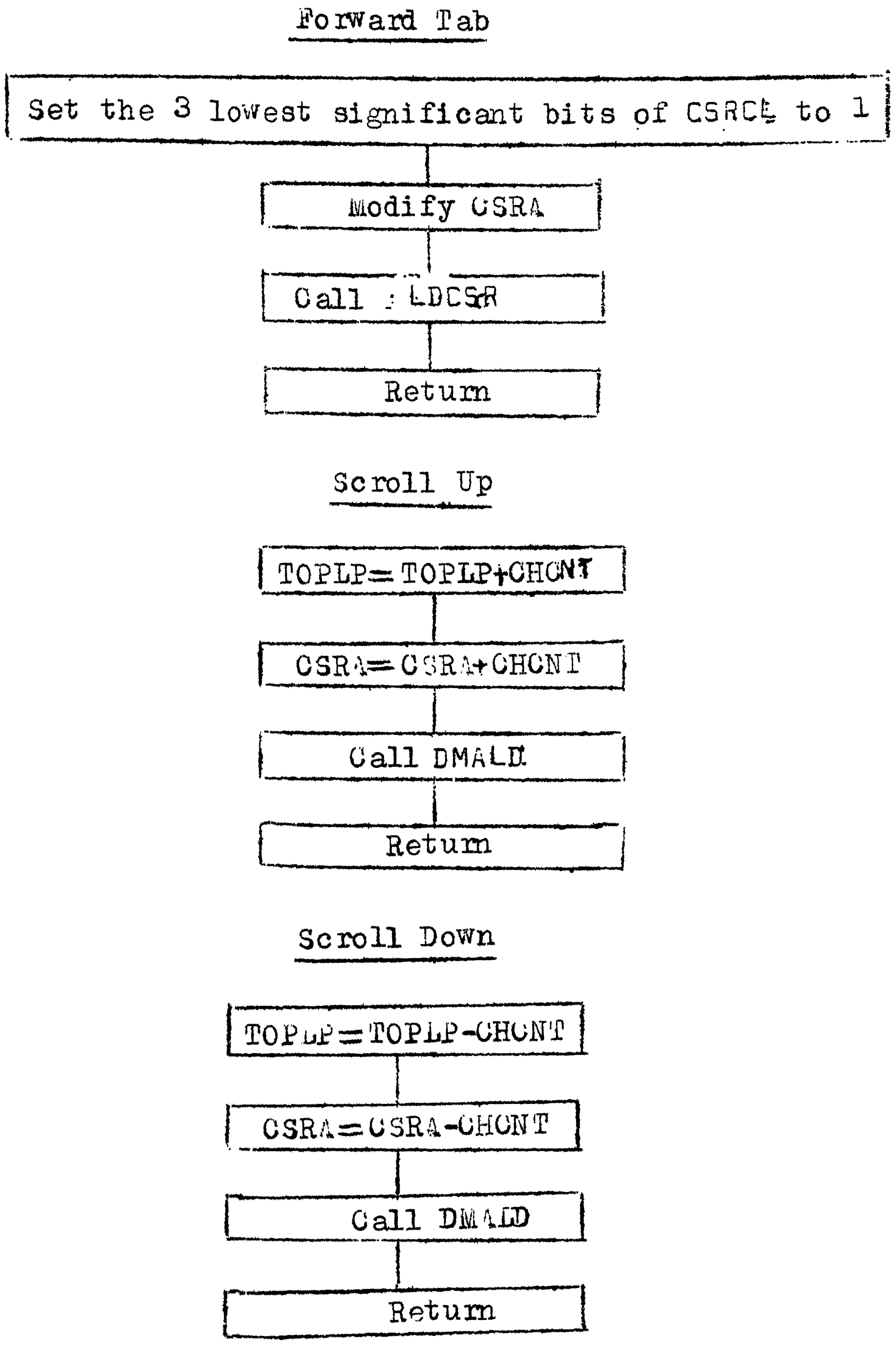

Forward Tab
Set the 3 lowest significant bits of CSRCL to 1
Modify CSRA
Call : LDCSR
Return
Scroll Up
TOPLP= TOPLP+CHCNT
CSRA=CSRA+CHCNT
Call DMALD
Return
Scroll Down
TOPLP=TOPLP-CHCNT
CSRA=CSRA-CHCNT
Call DMALD
Return

FIG. 4.7

Skip routine is a special case of scroll-up routine. In this case screen is scrolled up by a number of lines and not just by one. This number is user definable. Flow chart is given in Fig. 4.8.

4.4 STORAGE OF KEYED-IN CHARACTER

This category of routines is invoked when a character is to be stored into memory. It may be an ordinary character or a character in the protected field. The routines which fall in this category are as follows:

(1) Start Protection

(2) Delimit Protection

(3) Remove Protection

(4) Store Character

(5) Protection Check

First 3 routines are quite simple and their flow charts, which are self explanatory, are given in Fig. 4.9.

Store character routine first of all checks whether memory reference is legal or not i.e. whether the user is accessing only the allowed memory area or not. If memory reference is not illegal, it calls Check Protection routine and then stores the character depending on the information provided by the check protection

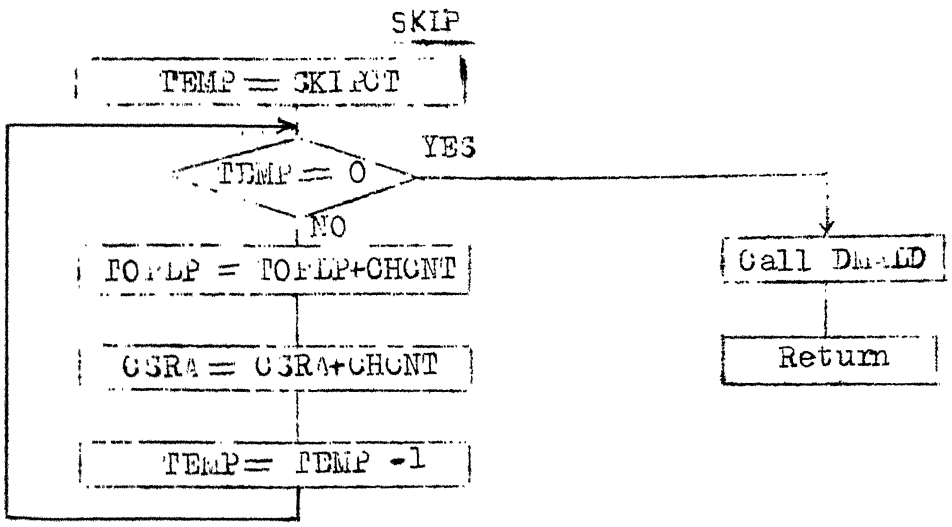
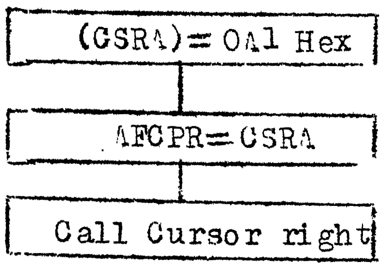

SKIP
TEMP = SKIPCT
TEMP = 0
YES
NO
TOPLP = TOPLP+CHCNT
CSRA = CSRA+CHCNT
TEMP = TEMP -1
Call DMALD
Return

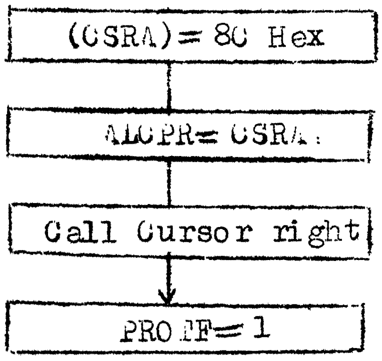
Fig. 4.8

Start Protection



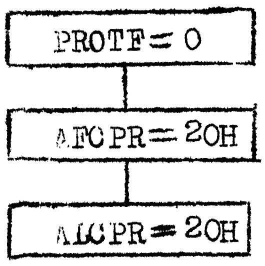
Note: 0A1 code when supplied to CRT Controller makes the text following highlighted with an underline.

Delimit Protection



Note: 80 code eliminates the effect of highlighted with underline attribute.

Remove Protection



Note: 20 code has been filled in place of 0A1 and 80 which were filled while setting and delimiting the protected region.

FIG. 4.9

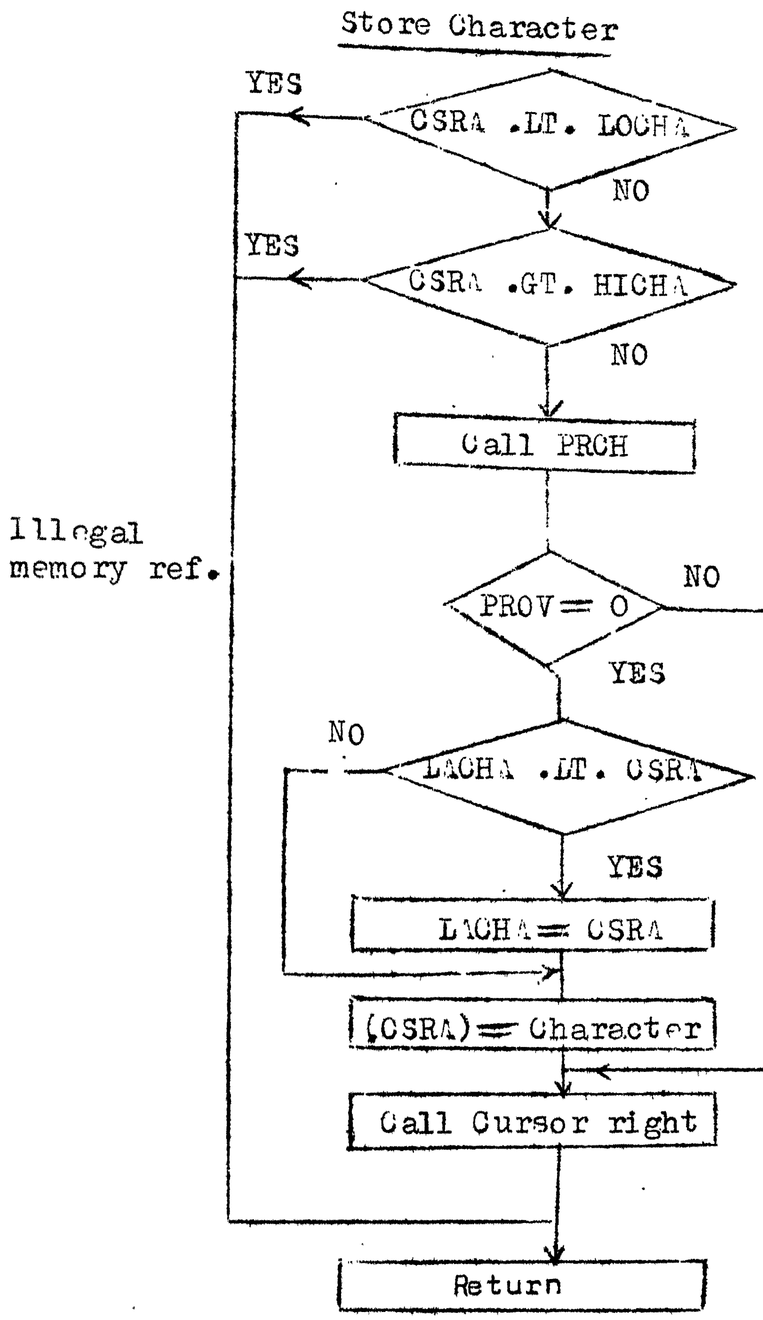

Store Character
YES
CSRA .LT. LOCHA
NO
YES
CSRA .GT. HICHA
NO
Call PRCH
Illegal memory ref.
PROV = 0
NO
YES
NO
LACHA .LT. CSRA
YES
LACHA = CSRA
(CSRA) = Character
Call Cursor right
Return

FIG. 4.10

routine. The flow chart is given in Fig. 4.10.

Protection Check routine is called either by the TRAP routine if a character is to be stored in memory or by the insert or delete routines, invoked as a Command within the TRAP routine, to ensure that no modification is made in the Protected region, if any. This routines update either NOIOF or PROVF depending upon the invoking source and the situation involved. The flow chart is given in Fig. 4.11. This Fig. contains the flow chart of Clear Memory Routine also which is invoked by the user when he wants to clear a certain area of RAM, usually before entering his program.

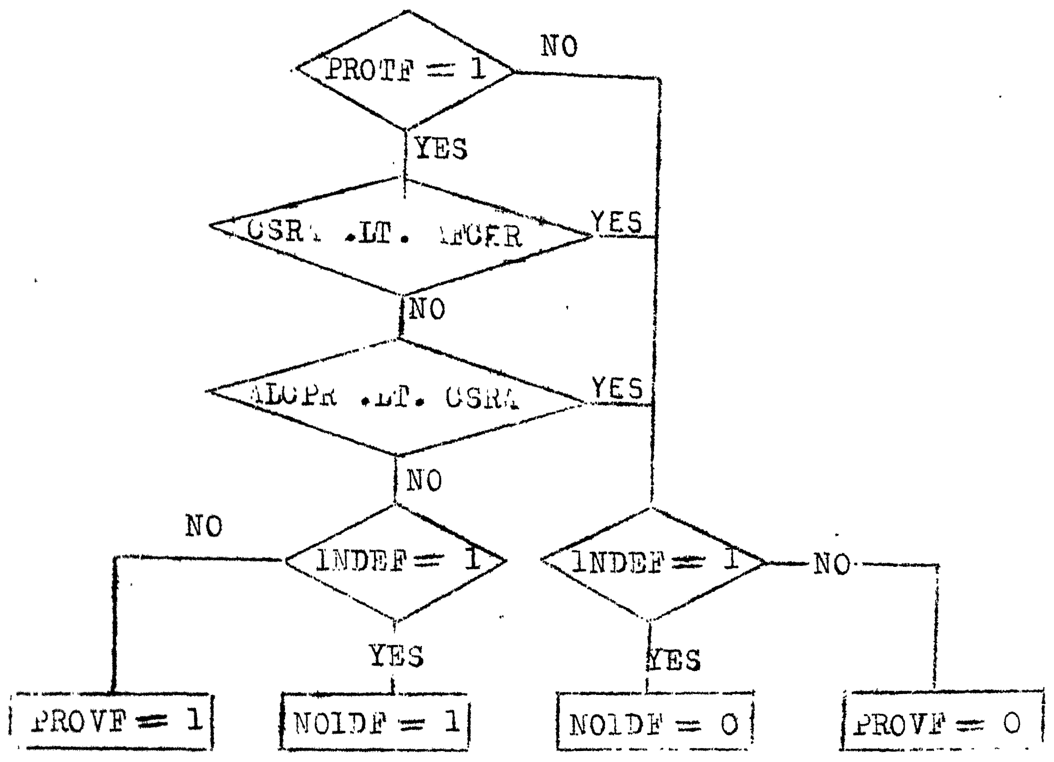
4.5 EDITING COMMANDS

The edit command which have been discussed here are as follows :

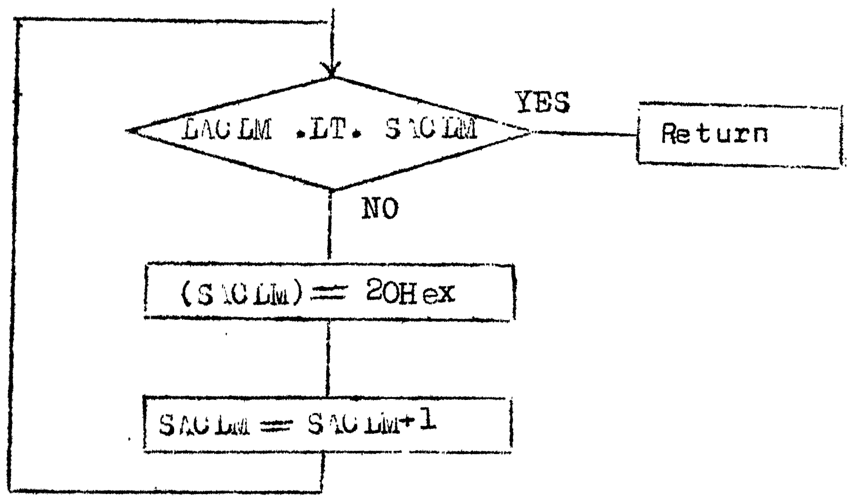
(1) Insert Character
(2) Insert Line
(3) Delete Character
(4) Delete Line

All these 4 routines use 2 flags INDEF and NOIDF. INDEF = 1 indicates that some insert or delete routine is in progress. This flag is used by the protection check routine (invoked by these four routines)

Check Protection



Clear Memory Routine



Note: 20 Hex is ASCII Code of Space.

FIG. 4.11

to modify the value of NOIDF depending upon whether protection is violated. Before commencing individual function, these routines check the flag NOIDF and if set, do nothing, otherwise execute the full routine to give the desired result. Finally INDEF is reset.

Insert character routine increments by one the absolute addresses of all characters following the cursor so that these characters appear in the next to present column. Character displayed in cursor position is also moved to the right by one column and then Blank is inserted in the cursor position. The flow chart of this routine is given in Fig. 4.12.

Insert line routine increments the absolute addresses of all characters falling in the cursor row and the subsequent rows by the number of characters per row so that each character in cursor row or subsequent rows appear in the next to present row. It inserts blanks in the cursor row. Cursor is moved to the first character position of the cursor row. The flow chart is shown in Fig. 4.13.

Delete character routine decrements by one the absolute addresses of all characters following the cursor so that they appear in positions one column left to their recent positions. Cursor character is automatically

INSERT CHARACTER

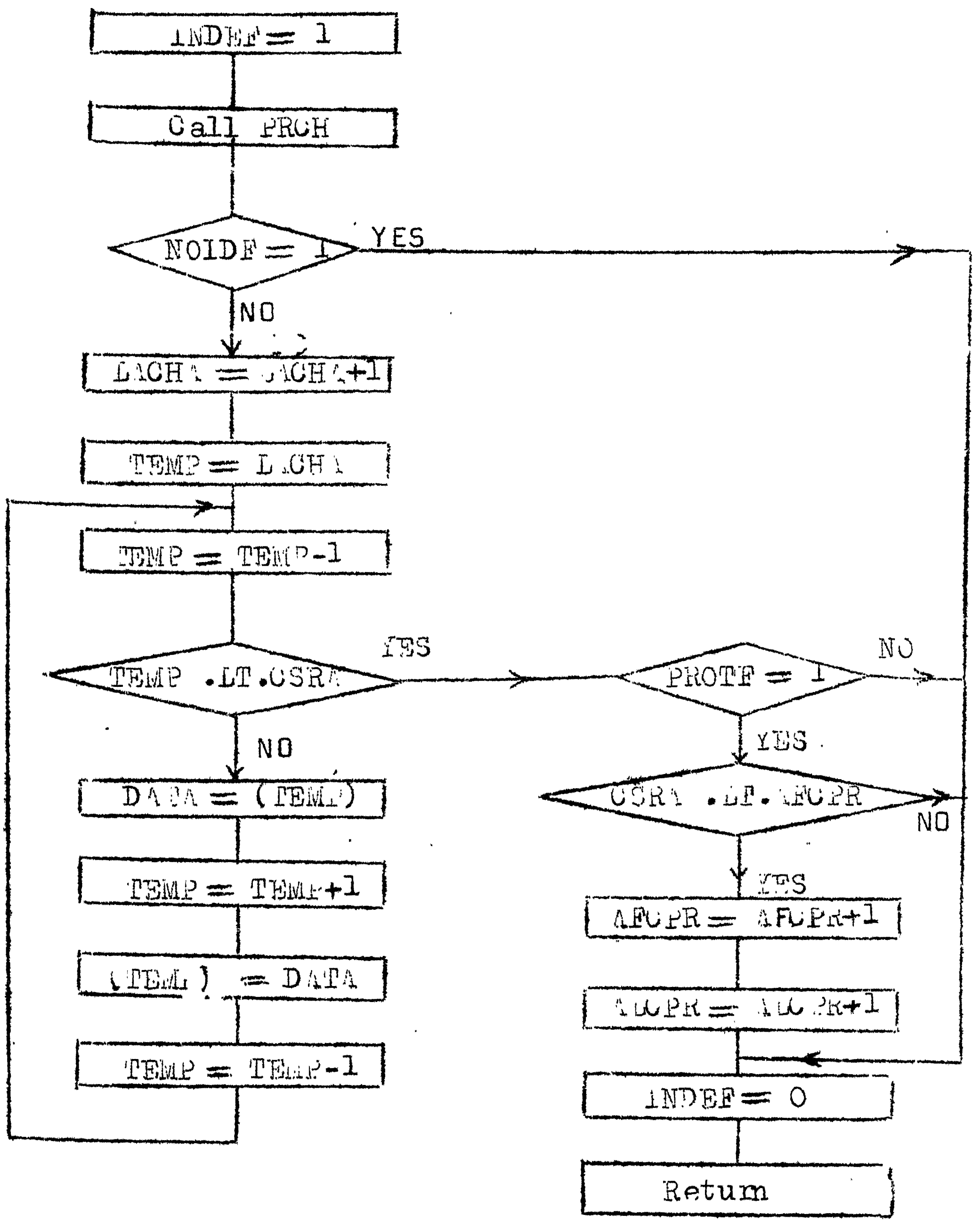


FIG.4.12

Insert Line

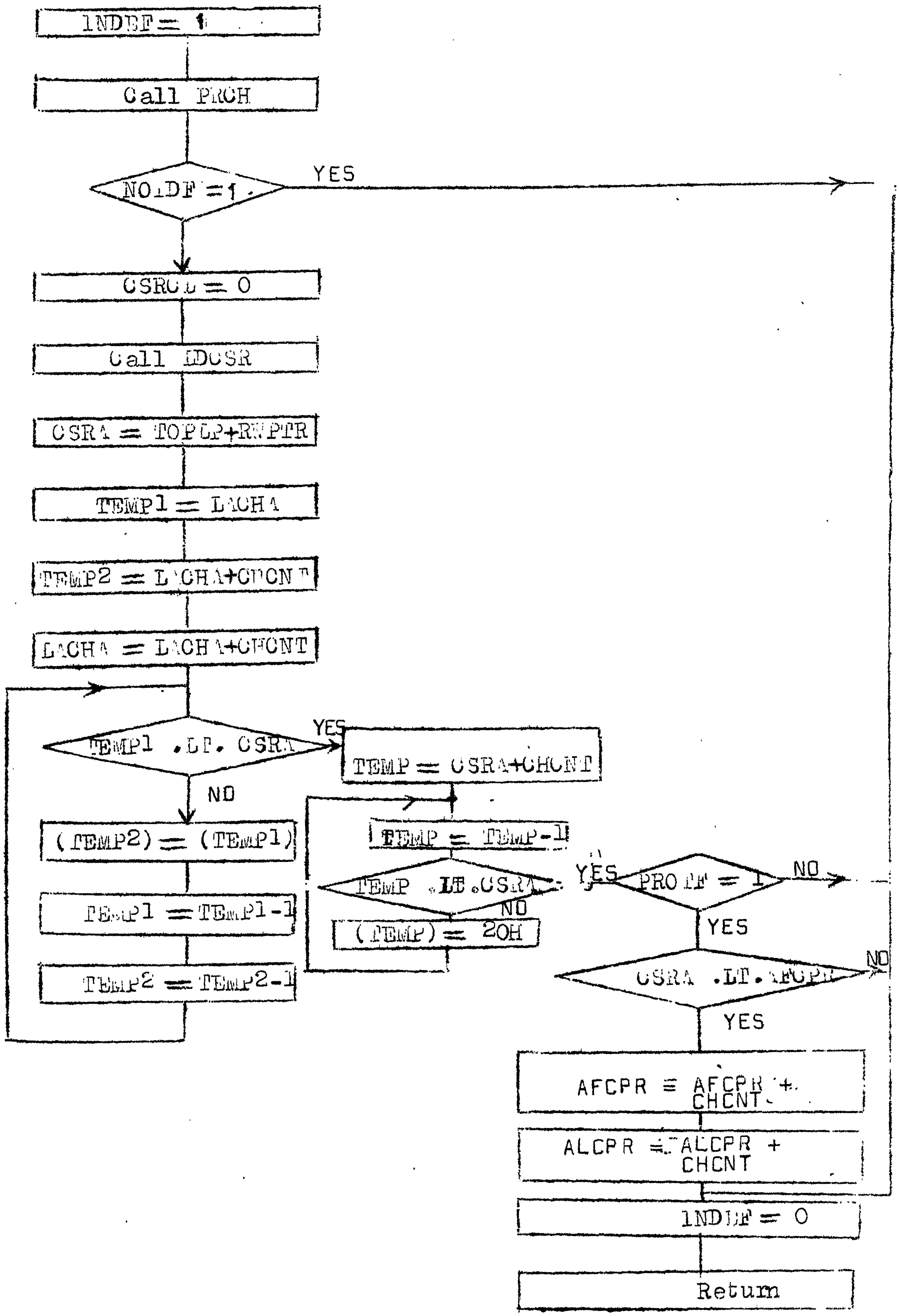


FIG.4.13

DELETE CHARACTER

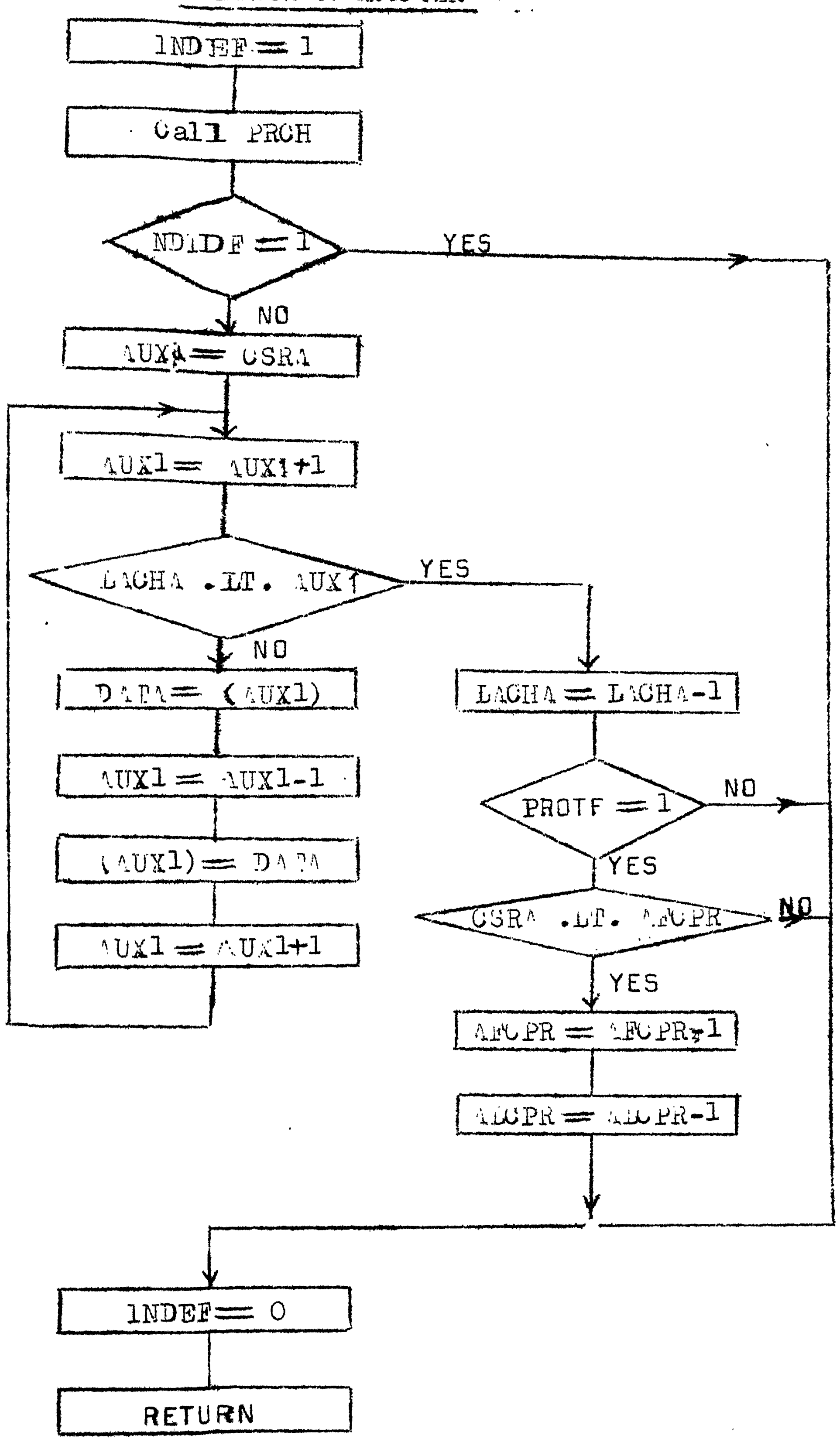


FIG.4.14

Delete Line

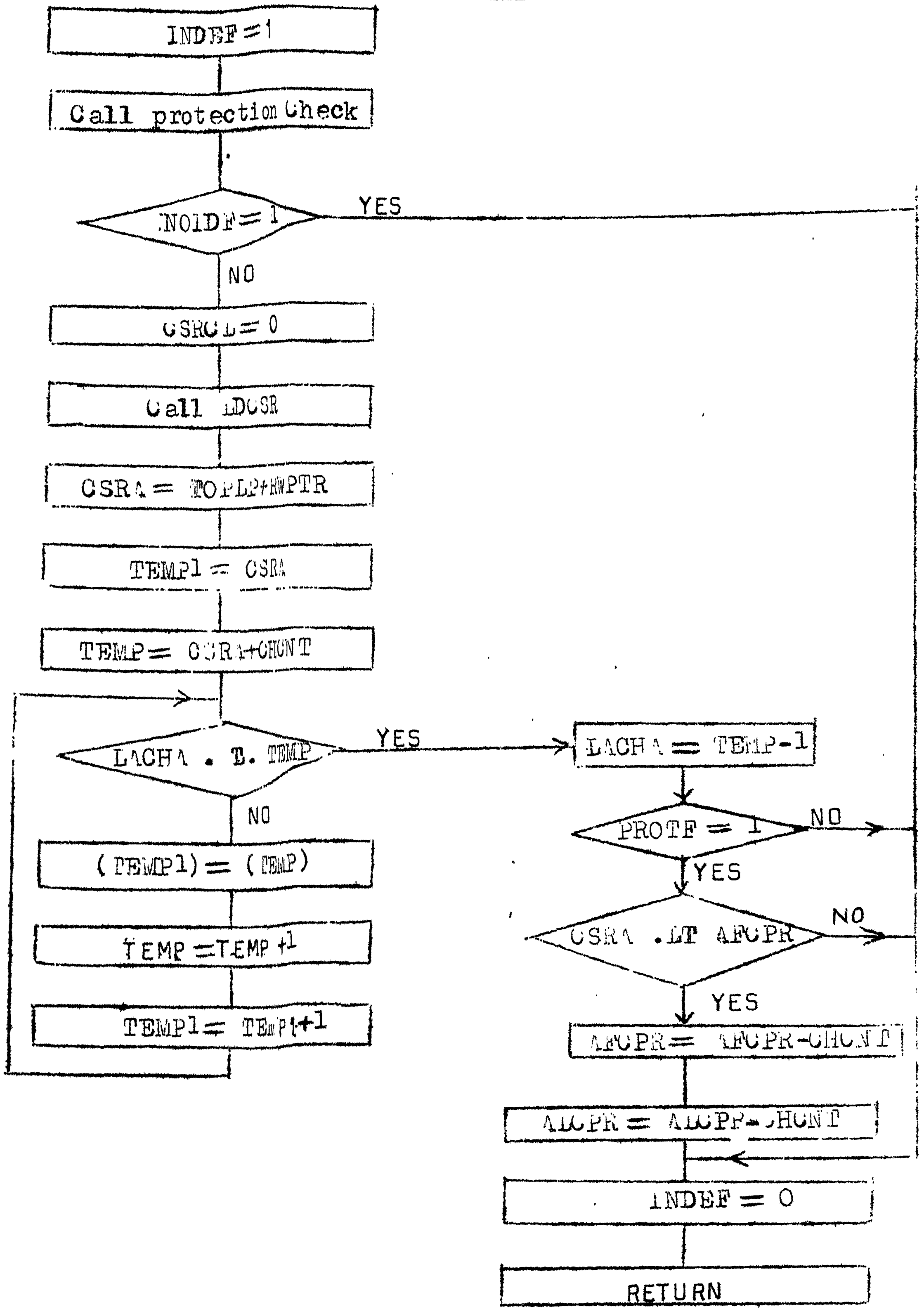


Fig. 4.15

deleted. The flow chart is given in Fig. 4.14.

Delete line routine decrements the absolute addresses of all characters falling in the subsequent rows of cursor row by the numbers of characters per row so that the characters in the cursor row are deleted and the characters following the cursor row are shifted one line up. Cursor appears in the first column of the cursor row. The flow chart is given in Fig. 4.15.

All the four routines modify the addresses of the protected region if these are changed during the execution of the routines.

4.6 SPECIAL COMMANDS

The various special commands are discussed one by one in this section:

4.6.1 Parameters Store Command :

When the user desires to use this facility, he connects RST 7.5 interrupt of 8085 to STB input of keyboard. (This interrupt line is usually connected to USART). To change the parameter the user has to specify the lower byte of the address of the parameter and the value of the parameter. Each byte (address/

Parameters Store

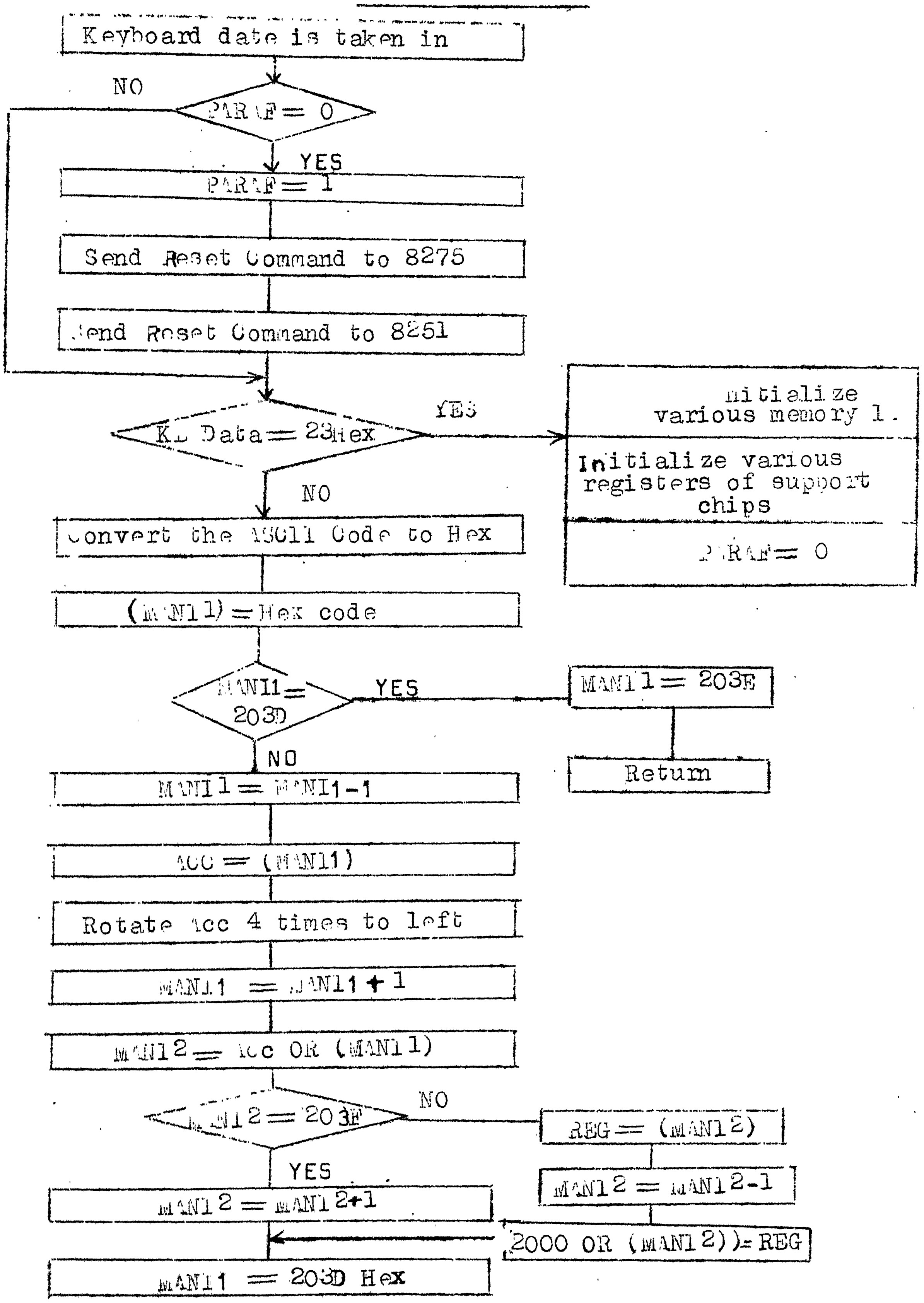


Fig. 4.16

value) is keyed in Hex so the user presses 2 keys for each byte to be entered (e.g. if the desired byte is A2, the user presses key A and then 2). The converted codes of each of these 2 keys are stored temporarily in a location pointed by MANI1. MANI1 contains either 203D or 203E. It is initialized to 203D by the main routine. So when the user presses the first key, its code goes to 203D. Having stored the code, MANI1 content is changed to 203E so the next key code is stored in 203E. Again, after this it is changed to 203D. Thus the content of MANI1 keeps on changing between 203D and 203E; 203D contains those 4 bits (with leading zeros) which are to be placed in the higher order 4 bit of the byte to be derived,while 203E contains those 4 bits (with leading zeros) which are to be placed in the lower order 4 bits of the byte to be derived.

When the code is received in 203E, content of 203D and 203E are grouped to make a single byte to be derived. This byte is stored in a location pointed by MANI2. MANI2 is initialized to 203F. As lower byte of the address of the parameter is specified first, it is stored in 203F. After this MANI2 is changed to 2040 so the value of the parameter (keyed-in next) is stored in 2040. Again MANI2 is changed to 203F. Once 2040 is filled, this value is stored in the appropriate memory location (higher byte of address is 20 Hex as pointed out in section 4.1 and lower byte of address is specified by the location 203F). Thus lower address byte of any parameter is stored in 203F while the value of the parameter is in 2040.

As no other activity in the system should run while the parameters are being changed, the routine sends reset command to CRT Controller and disables USART as soon as this routine is initiated. This is accomplished by using a PARAF as shown in the flow chart given in Fig. 4.16.

When all the parameters have been changed, the user keys in '.' key which is an indication to the system that all parameters to be changed have been entered. (At this time the USER connects RST 7.5 interrupt of 8085 back to USART) Upon detecting '.', CPU invokes a routine MANI which initializes the various support devices and some memory locations in the same way as the main routine does.

4.6.2 DUMP COMMAND

It is used to transfer blocks of data from one area of the memory to another. This is useful in programming a new ROM (in which case RAM content is written in ROM) or in reprogramming an already pro-

DUMP Routine

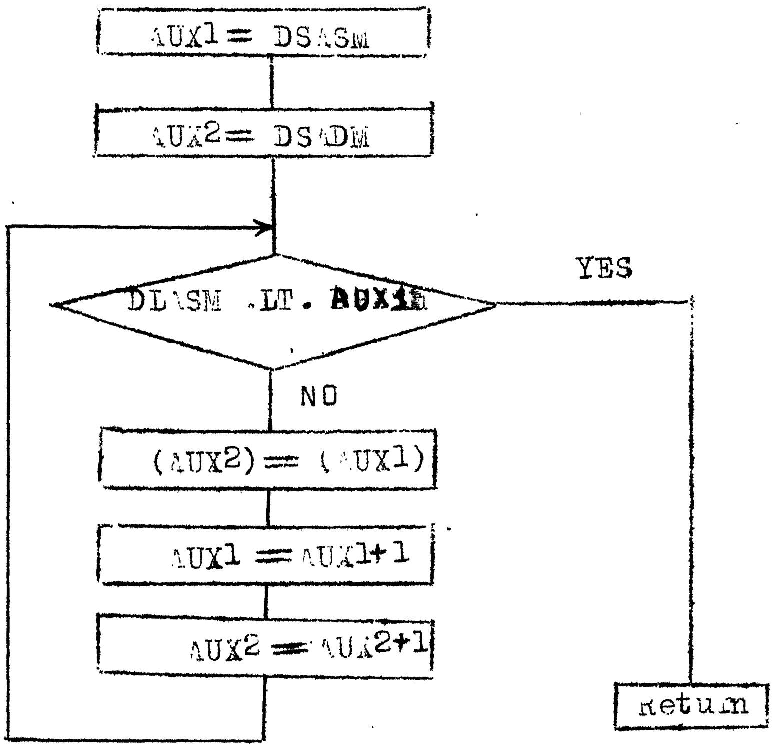


FIG.4.17

grammed ROM. (in which case firstly ROM content is transferred to RAM, modified there and then written into ROM). The user specifies source addresses and the destination address. The flow chart is shown in Fig. 4.17.

4.6.3 ASCII TO HEX CONVERSION

This routine is required when a text entered through keyboard is to be executed or certain commands are to be given directly from the keyboard to the associated peripherals or an EPROM is to be programmed (content of its various locations are sent through the keyboard). In such a case ASCII code is to be converted to Hex. For this the user invokes this routine. Before invoking, the user again specifies the source addresses (first and last address of ASCII text) and destination address (first address where the converted code is to be stored). The complete flow chart is shown in Fig. 4.18.

4.6.4 Hex to ASCII CONVERSION

This routine may be required while reprogramming an EPROM (In such a case need arises for displaying the content of ROM, which is nondisplayable, so that the user may easily modify the various locations) or while testing LSI's (in which case again

ASCII TO HEX CONVERSION

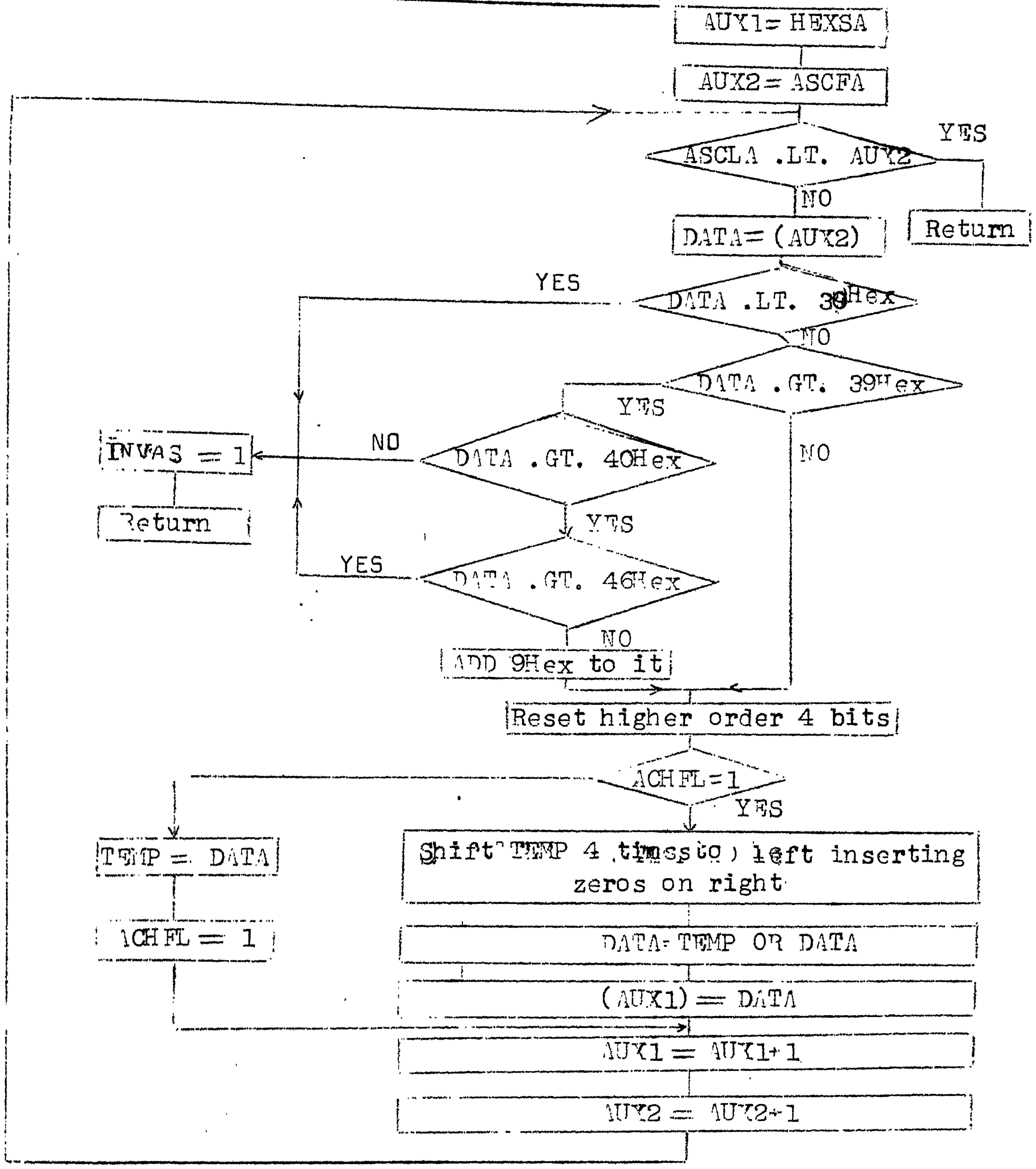


Fig. 4.18

HEX TO ASCII CONVERSION

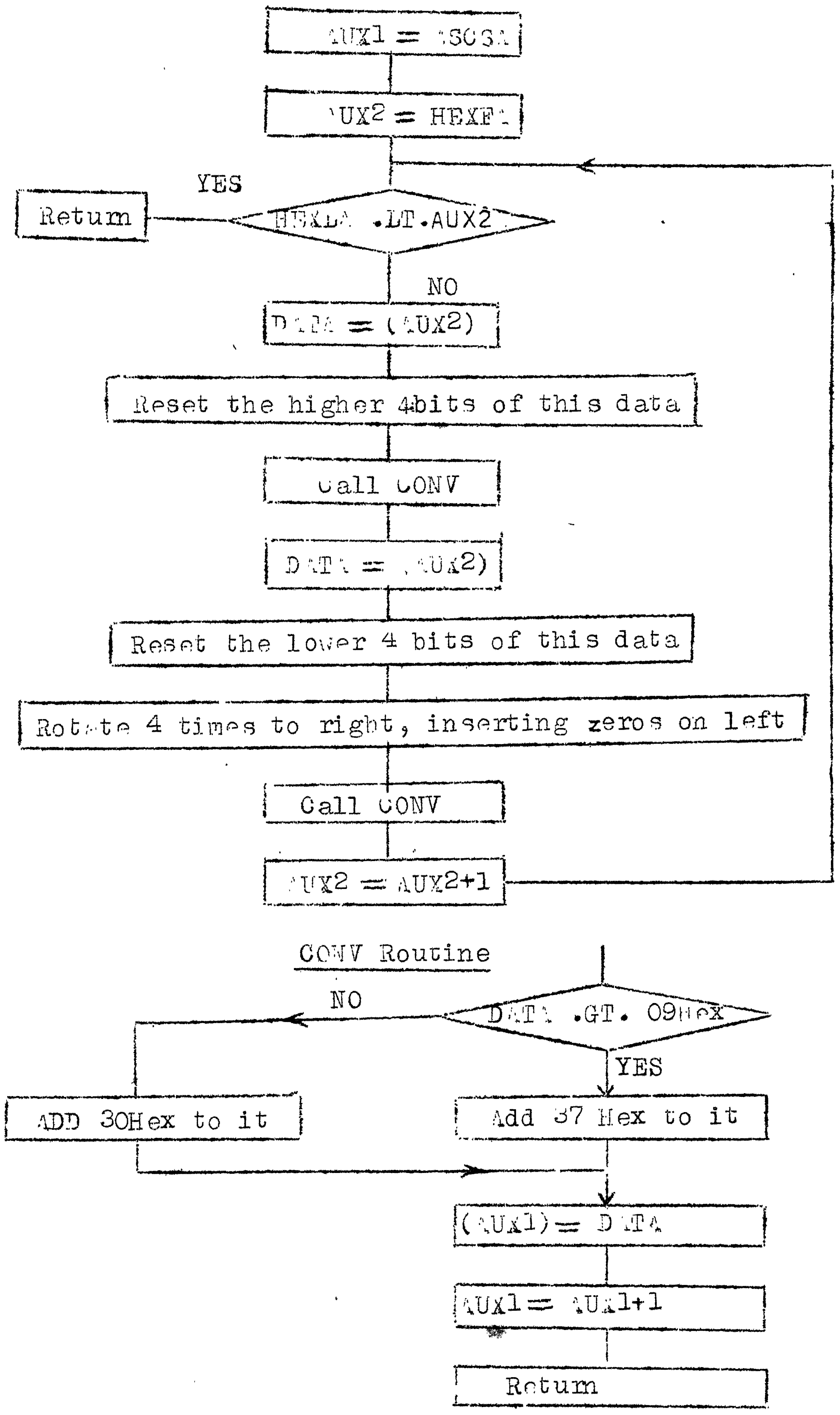


FIG. 4.19

the response of the chips is non-displayable). This routine converts the non-displayable code (which is executable only) to displayable one. For this routine again source and destination address are required. The complete scheme is shown in Fig. 4.19.

4.6.5 Bit set reset Command

This command sets/resets the specified bit of port C of 8255 to suit the user's requirements.

4.6.6 Set Ecape Flag Command

This command sets the escape flag which is an indication that the following key will represent a character attribute. This is useful in generating graphics.

CHAPTER 5

CONCLUSION

The objective of the project was to develop a low-cost intelligent CRT terminal to suit the academic requirements where a user can edit a text, run his machine-language program, transmit/receive some text on a serial line etc. The design was done using a Direct Memory Access chip considering the fact that the CPU should be relieved of data transfer so that it can be devoted more usefully for some decision making such as program execution.

But as the DMA chip did not function properly, the possibility of avoiding DMA was thought of. This time more critical study of time saving was done and it was found that instead of using a conventional software routine, which has simple memory fetching instructions, incrementing the address, sending the content of these locations to the CRT controller and taking some critical action depending upon the address (as described in Chapter 3), if POP technique is used, the net time available to the CPU for decision making is not significantly lower than that in a DMA based system.

One POP instruction transfers two bytes of data to the CRT controller and takes only 10 clock cycles to achieve it i.e. for a net transfer of one byte, the POP technique just takes five clock cycles. On the other hand DMA takes 4 clock cycles to achieve net transfer of one data byte. Moreover some time is further needed to achieve proper handshake between CPU and DMA. So the difference between the two scheme is one clock cycle per byte maximum, which is not worthwhile considering the cost of a DMA chip as compared to that of the CPU. In fact, one can even think of putting a dedicated processor for display alone as it would result in cost reduction as well as increased time available for decision making. Though, of course, overheads increase because of the need of proper control of 2 processors.

However in view of the fact that even a system based on single CPU can provide sufficient time for execution of simple programs as envisaged in our original requirements, the software was finally tested out on such a system. The actual hardware is essentially a simplification of the schematic described in Chapter 3 obtained by eleminating DMA and putting a simple decoding scheme which converts processor READ signals, during POP instructions, to CRT controller WRITE and DACK signals to simulate the DMA chip. The complete software has been successfully tried out on such a system.

APPENDIX A

Special Commands	Associated Key
1. CURSOR HOME	ESC Q
2. CURSOR LEFT	ESC H
3. CURSOR RIGHT	ESC R
4. CURSOR UP	ESC U
5. FORWARD TAB	ESC F
6. CARRIAGE RETURN WITH LINE FEED	ESC M
7. LINE FEED	ESC J
8. SCROLL UP	ESC P
9. SCROLL DOWN	ESC N
10. SKIP	ESC S
11. INSERT CHARACTER	ESC I
12. DELETE CHARACTER	ESC D
13. INSERT LINE	ESC L
14. DELETE LINE	ESC C
15. INSERT CHARACTER (WRAP AROUND)	ESC <
16. DELETE CHARACTER (")	ESC >
17. INSERT CHARACTER STRING IN A LINE	ESC ;
18. SET PROTECTION	ESC V
19. DELIMIT PROTECTION	ESC W
20. REMOVE PROTECTION	ESC X
21. HEX TO ASCII CONVERSION	ESC B

Special Commands	Associated Key
22. ASCII TO HEX CONVERSION	ESC A
23. USER'S PROGRAM EXECUTION	ESC E
24. ERASE MEMORY	ESC K
25. MEMORY TRANSFER	ESC T
26. CHANGING CERTAIN PARAMETERS	ESC Q
27. 8255 PORT BIT SET RESET	ESC =
28. START TRANSMISSION	ESC Y
29. STOP TRANSMISSION	ESC ?
30. START RECEPTION	ESC Z
31. STOP RECEPTION	ESC @
32. ASCII TEXT ADDRESS CALCULATION	ESC :

APPENDIX B

ATTRIBUTES OF CRT CONTROLLER

Graphic Generated	Description	Associated key	Appearance
┌	Top left corner	Shift 2	No Blink, No Highlight
		A	Highlight
		B	Blink
		C	Blink, Highlight
┐	Top right corner	D	No Blink,No Highlight
		E	Highlight
		F	Blink
		G	Blink, Highlight
└	Bottom left corner	H	No Blink,No Highlight
		I	Highlight
		J	Blink
		K	Blink, Highlight
┐	Bottom right corner	L	No Blink,No Highlight
		M	Highlight
		N	Blink
		O	Blink, Highlight
┬	Top intersect	P	No Blink,No Highlight
		Q	Highlight
		R	Blink
		S	Blink, Highlight
┤	Right intersect	T	No Blink,No Highlight
		U	Highlight
		V	Blink
		W	Blink, Highlight
├	Left intersect	X	No Blink,No Highlight
		Y	Highlight
		Z	Blink
		[	Blink, Highlight
┴	Bottom intersect	/	No Blink,No Highlight
		]	Highlight
		Shift 6	Blink
		Shift -	Blink, Highlight

Graphic Generated	Description	Associated key	Appearance
─	Horizontal line	'	No Blink, No Highlight
		Shift A	Highlight
		Shift B	Blink
		Shift C	Blink, Highlight
│	Vertical line	Shift D	No Blink, No Highlight
		Shift E	Highlight
		Shift F	Blink
		Shift G	Blink, Highlight
┼	Crossed lines	Shift H	No Blink, No Highlight
		Shift I	Highlight
		Shift J	Blink
		Shift K	Blink, Highlight

SPECIAL CODES OF 8275

Code	Associated key
End of Row	Shift P
End of Row, Stop DMA	Shift Q
End of Screen	Shift R
End of Screen, Stop DMA	Shift S

Note: These attributes are keyed-in by pressing Escape Key followed by the associated key as mentioned above.

ÈE-1981-M-MAY-INT